VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce chodců na embedded systémech

Pedestrian Detection on Embedded Systems

2018 Jakub Ševčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Ševčík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce chodců na embedded systémech
Pedestrian Detection on Embedded Systems

Jazyk vypracování: čeština

Zásady pro vypracování:

Tato práce je zaměřena na prozkoumání oblasti detekování chodců na embedded systémech. Kombinace vysokého výkonu těchto systémů s možností dalšího rozšíření nabízí ideální řešení například pro automobilové a průmyslové využití.

1. Seznamte se se základními pojmy v oblasti detekce chodců v obrazech.
2. Prozkoumejte, jaké možnosti nabízí dostupné knihovny.
3. Implementujte detektor chodců, který poběží na embedded systémech (využít můžete výše nastudované knihovny).
4. Pokuste se detekci urychlit pomocí algoritmů, které analyzují pohyb objektů v obrazech.
5. Experimentálně ověřte funkčnost, přesnost a rychlost detektoru. Své závěry řádně zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] Navneet Dalal and Bill Triggs: Histograms of Oriented Gradients for Human Detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 1. IEEE Computer Society, USA, pp. 886-893, 2005
[2] Javier Marín, David Vázquez, Antonio M. López, Jaume Amores, and Bastian Leibe: Random Forests of Local Experts for Pedestrian Detection. In Proceedings of the 2013 IEEE International Conference on Computer Vision (ICCV). IEEE Computer Society, USA, pp. 2592-2599, 2013

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radovan Fusek, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Na tomto místě bych rád poděkoval Ing. Radovanu Fuskovi, Ph.D. za jeho odborné rady, trpělivost a ochotu se mnou konzultovat veškeré problémy. Dále bych rád poděkoval svým přátelům a rodině za psychickou podporu. Bez nich by tato práce určitě nevznikla.

Abstrakt

Detekce chodců v posledních letech přitahuje velkou pozornost, a to hlavně v rámci bezpečnostních aplikacích. Například jí lze využít v bezpečnostních kamerách na veřejně dostupných místech nebo v automobilových systémech, které díky tomu mohou zabránit nebezpečí. Pro detekci se dají použít čím dál rozšířenější embedded systémy, jako je například Raspberry PI. Jsou to malé a dobře dostupné počítače, které mohou být umístěné takřka kdekoliv. Primárním cílem práce je otestovat a pokusit se optimalizovat rozpoznávací techniky pro taková zařízení. Detekci chodců v obrazech řeší například histogram orientovaných gradientů, zkráceně HOG. Optimalizaci algoritmu zajišťuje substrakce pozadí, která tento proces značně urychluje. V rámci této práce jsou tyto algoritmy popsány. Dále zde budou popsány další možné dosavadní techniky detekce chodců v obrazech. Součástí práce je také srovnání úspěšnosti a rychlosti detektoru na vybraných zařízeních.

Klíčová slova: detekce chodců, embbeded systémy, optimalizace, substrakce pozadí

Abstract

Pedestrian detection has attracted great attention in recent years, especially in safety applications. For example, it can be used in security cameras in public places or in automotive systems, which can prevent potential accidents. For detection, embedded systems such as Raspberry PI can be used. These are small and affordable computers that can be placed almost anywhere. The goal of the thesis is to test and attempt to optimize the recognition techniques for such devices. Pedestrian detection in images processing is solved, for example, by Histograms of Oriented Gradients, abbreviated as HOG. Optimization of the algorithm is ensures by background subtraction, which greatly accelerates detection process. These algorithms are throughly described in this thesis. Further pedestrian detection techniques will be described her. Part of the thesis is also a comparison of detector rate and speed on selected devices.

Key Words: Pedestrian Detection, Embedded Systems, optimization, background subtraction

Obsah

Seznam použitých zkratek a symbolů

AUC	–	Area Under Curve
ALG	–	Algoritmus
API	–	Application Programming Interface - rozhraní pro programování aplikací
ARM	–	Advanced RISC Machine - architektura počítačů s nízkou elektrickou spotřebou energie
C++	–	Programovací jazyk C Plus Plus
CPU	–	Central Processing Unit - Centrální procesorová jednotka
Dlib	–	Otevřená cross-platform knihovna pro zpracování obrazu a strojového učení
FPS	–	Frames per second - Počet snímků za sekundu
FPR	–	False Positie Rate
IoT	–	Internet of Things - Internet věcí
MMX	–	Multi Media Extension, multimediální technologie vytvořené firmou Intel
NEON	–	Pokročilé rozšíření architektury SIMD pro procesory ARM Cortex-A a Cortex-R52
OpenCV	–	Otevřená knihovna pro zpracování obrazu a strojového učení
RGB	–	Red Green Blue - barevný prostor složený ze tří barevných kanálů
ROC	–	Receiver operating characteristic
SIMD	–	Single Instruction Multiple Data - typ počítačové architektury
SSE	–	Streaming SIMD Extensions je instrukční sada typu SIMD
SVM	–	Support vector machines
TPR	–	True Positive Rate
XML	–	Extensible Markup Language
YML	–	YAML Ain't Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Detekce chodců představuje velmi náročný úkol, který v posledních letech přitahuje velkou pozornost. Aplikace tohoto typu mohou mít široké uplatnění jak v osobním, tak v industriálním využití. Může se jednat o bezpečnostní prvky, například na letištích, kde program může sledovat pohyb daného chodce a vyhodnocovat tak jeho chování. Případně jako kamera průmyslového vozidla, kde řidič může přehlédnout chodce v blízkosti vozidla, a předejít tak neštěstí, kdy program může zamezit pohybu vozidla. Dalšími příklady mohou být detekce chodců na přechodech pro chodce, v továrnách, kde se může pomocí rozšíření algoritmu o rozpoznání obrazu vyhodnocovat chování a docházka zaměstnanců.

Nutno podotknout, že detekce chodců v obrazech není pro lidské oko tak obtížným úkolem a dokážou bez větší námahy rozpoznat všechny osoby v obraze. Pro stroje je naopak tento úkol velkou výzvou. Chodci v obrazech mohou mít různý tvar těla, barvu kůže, jiný postoj, také různý počet vrstev a barev oblečení na sobě. Mohou být také z části zakrytí nějakým objektem v obraze, který nepodléhá samotné detekci, což může způsobit záporné vyhodnocení. Také zde má velký vliv vzdálenost osoby od kamery, kdy se pro strojovou doménu může stát chodec nedetekovatelný.

Velkou zásluhu na strojovém detekování chodců má Navneet Dalal a Bill Triggs, kde ve své práci [6] pomocí metodiky histogramu orientovaných gradientů úspěšně detekují chodce v obrazech. Tato práce se z větší části inspiruje tímto dokumentem a rozšiřuje jej o substrakci pozadí, která má za úkol urychlit a zlepšit samotnou detekci chodců v obrazech. Principem tohoto algoritmu je spustit detekci pouze v oblastech obrazu, které se v čase mění a mohou představovat chodce. Ovšem tento typ algoritmu lze použít pouze na videosnímcích, které jsou pořízené ze statické kamery.

Hlavním cílem této práce je optimalizovat a otestovat detektor chodců pro počítače s architekturou ARM a využít tak jejich značný výkon. Experimenty této práce se zaměří jak na trénování a testování klasifikátoru s různými parametry, tak na detekci chodců na samostatných snímcích, ale také i ve videosekvencích, které budou detailně zaznamenány a vyhodnoceny. V druhé kapitole budou představeny hlavní výzvy a problémy detekce chodců v obrazech. Jedna z kapitol se bude věnovat metodikám pro detekci chodců. V dalších kapitolách bude popsána implementace programu, popis jeho funkcí, kterými tato aplikace disponuje a knihovny, které byly v této práci použity. Také zde budou uvedena a popsána zařízení, na kterých byl algoritmus otestován. V neposlední řádě experimenty a jejich dosažené výsledky.

2 Výzvy a problémy detekce chodců

Detekce chodců je nezbytným a významným úkolem v jakémkoliv inteligentním kamerovém systému, jelikož poskytuje základní informace pro sémantické porozumění videosekvence. Dále má zásadní význam v automobilovém průmyslu z důvodu možného zlepšení bezpečnostních systémů. Mnoho výrobců aut již tuto funkcionalitu nabízí od roku 2017 ve svých vozech jako „Pokročilé systémy asistence řidiče“ (ADAS)[3].

Bohužel ne vždy detektor rozpozná chodce v obraze, a to může být ovlivněno několika faktory. Tyto faktory, které ovlivňují detekční úspěšnost, jsou rozebrány v následujících sekcích.

2.1 Tvar těla (póza, držení těla)

Chodec v obraze může mít neobvyklý postoj, chůzi nebo různé proporce a stavbu těla. Lidské tělo může být zčásti zakryto nějakým objektem ze scény nebo se nemusí vyskytovat celé v záběru snímku. Takový člověk může být snadno identifikovatelný pro lidské oko, ale pro stroje může představovat velkou výzvu.

2.2 Barva kůže

Barva kůže může také představovat problém pro rozpoznání chodce. Na světě existují různé druhy pigmentace kůže a jejich odstíny se pohybují v rozmezí od nejtmavší hnědé až po nejsvětlejší odstíny. Příklady barev kůže ilustruje obrázek 1.

Obrázek 1: Příklady druhů pigmentace kůže[1]

2.3 Vnější podněty

Problém pro identifikování chodce v obraze může také představovat druh osvětlení scény, případně střídání dne a noci pro venkovní kamerové systémy. Také zde hraje důležitou roli nastavení a vlastnosti dané kamery.

2.4 Oblečení a různé doplňky

Do této kategorie spadá například zimní oblečení, které většinou zvětšuje objem či tvar těla, nebo různě rozměrné pokrývky hlavy. Chodec také může nést nějaké předměty, které mohou zakrývat jeho části těla nebo s ním dokonce splývat. Tyto aspekty také ovlivňují správnost detekce.

2.5 Pozice v obraze

Osoby se mohou nacházet libovolně v obraze, mohou jít čelem k ohnisku kamery nebo být zachyceny pouze z boku. Například kamera umístěná ve výšce pouličního osvětlení v nějakém předem určeném úhlu snímá obraz z určité perspektivy. Tato kamera pak zajišťuje velikou škálu chodců v různém měřítku. Pro stroj tohle může být problematické. Osoby ve větším měřítku mohou být snadno detekovatelné, ovšem chodci, kteří jsou hodně vzdálení v obraze, nikoliv. V tomto případě také záleží na nastavení daného detektoru za cenu pomalejšího výkonu, získáme přesnější detekci.

2.6 Pozadí (prostřední scény)

Chodci se většinou pohybují v komplexním venkovním prostředí. Opět, pro lidské oko může být člověk snadno identifikovatelný, a pro doménu strojů se může tato situace jevit jako neproveditelná. Osoba v obraze může totiž dokonale splynout s prostředním, které se za ní nachází.

Příklad výše zmíněných výzev a problémů detekce lze vidět na obrázku 2, který je umístěn níže. Na tomto ilustrativním obrázku najdeme osoby s různě barevným oblečením různého tvaru. Chodci se zde nacházejí v různém úhlu k ohnisku kamery.

Obrázek 2: Ukázka chodců v obraze

3 Metody detekce chodců

V této kapitole budou popsány nejpoužívanější a nejznámější detekční metodiky. Všechny implementace těchto metodik jsou dostupné minimálně v jedné knihovně použité v této práci. Prvně budou uvedeny metody založené na příznacích, jmenovitě Histogram orientovaných gradientů, Haar vlnky, Lokální binární vzory a některé jejich kombinace. Dále jsou zde uvedeny klasifikátory Support vector machine a AdaBoost. V poslední části této sekce se budu věnovat neuronovým sítím.

3.1 Metody založené na příznacích

Navzdory všem výzvám a problémům detekce chodců zůstává tato oblast výzkumu stále aktivní a byla navržena řada technik.

Histogram orientovaných gradientů

Tato metoda vznikla v roce 2005 za účelem detekování chodců v obrazech a je založena na histogramech orientovaných gradientů (*HOG - Histogram of Oriented Gradients*). Autoři této metodiky jsou N. Dalal a B. Triggs [6]. Základní myšlenka spočívá v tom, že okolní vzhled a tvar objektu může být často charakterizován distribucí intenzity gradientů nebo směry hran, aniž by byly přesně známy odpovídající gradientové nebo hranové polohy.

Před samotným výpočtem příznaků by mělo dojít k zajištění normalizaci barev a gamy, v případě černobílých obrázků k normalizaci kontrastu. Tento krok může být však vynechán, jak zdůrazňují autoři Dalal a Triggs. Normalizace deskriptorů dosahují stejného výsledku, a tedy předběžné zpracování obrázku má malý vliv na výkon. Místo toho je prvním krokem výpočet gradientních hodnot. Nejběžnější metodou je aplikování jednodimenzionální derivační masky v jednom nebo obou směrech, jak horizontálním, tak vertikálním. Autoři také experimentovali s komplexnějšími maskami, jako je například 3×3 Sobelova maska, nebo diagonální masky, avšak tyto masky se prokázaly jako méně účinné. Stejně tak neúčinné bylo použití jakéhokoliv vyhlazení obrazu před aplikací derivační masky. Autoři této metody přišli na to, že nejlepší kombinací je použití konvolucí Gaussovského filtrování $\sigma = 0$ na obraze I s maskou $[-1, 0, 1]$, $[-1, 0, 1]^\top$:

$$I_x = I \ * [-1, 0, 1],$$
$$I_y = I \ * [-1, 0, 1]^\top$$

kde:

$*$: konvoluce,

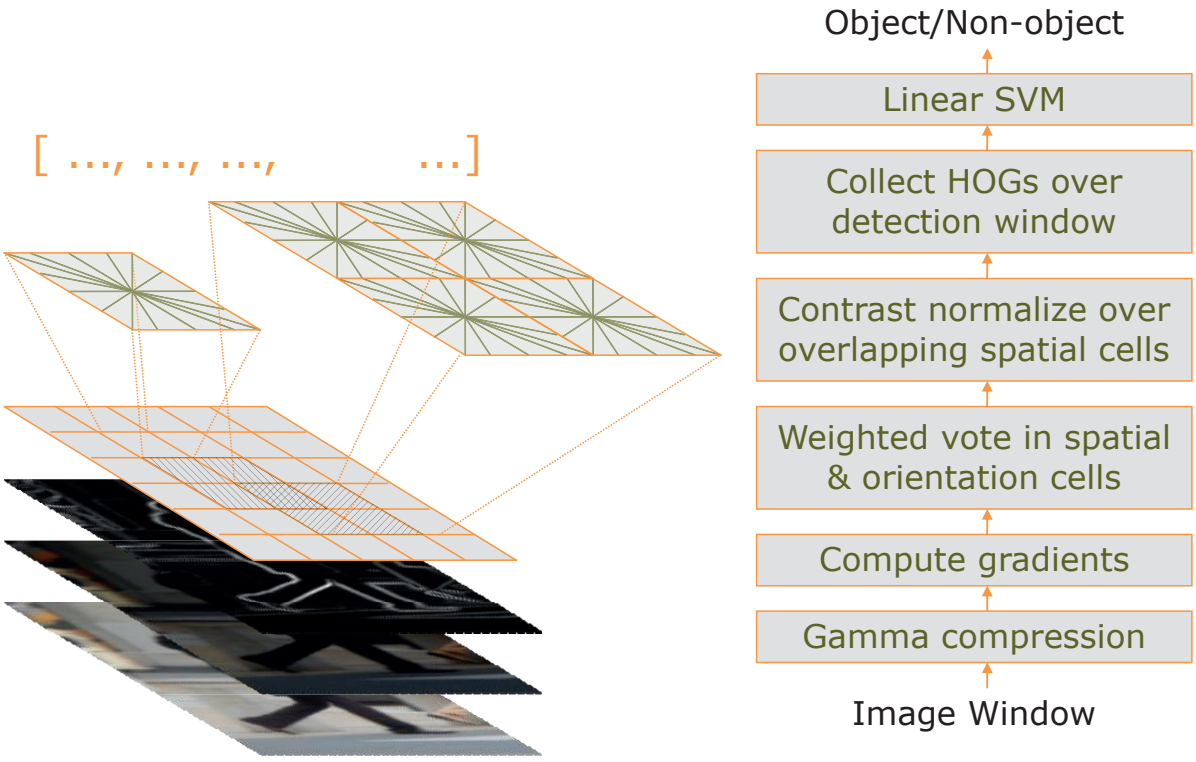
I: obraz.

V druhém kroku dochází k vytvoření histogramu v každé buňce, která je standardně tvořena 8×8 pixely. Každý pixel v buňce má svou váhu, kterou se podílí na vytvoření orientovaného

histogramu, založeného na hodnotách nalezených ve výpočtu gradientu. Hodnoty buněk jsou rovnoměrně rozloženy do histogramu o 9 kanálech (binech) po 20°. Pokud směr gradientu buňky vyjde na pomezí dvou binů, přičte se jeho magnituda do obou těchto binů. Tyto buňky spojíme do větších propojených bloků z důvodu normalizace osvětlení a kontrastu. Pro chodce se používá L2–norm normalizace, dle vztahu (1). Tyto bloky se typicky překrývají, což znamená, že každá buňka přispívá více než jednou do finálního deskriptoru. Proces zpracování deskriptoru je ilustrován na obrázku 3. Existují dvě varianty spojení bloků, tzv. obdélníkové bloky (R–HOG) a kruhové bloky (C–HOG). Tyto bloky jsou na obrázku 4.

$$\begin{aligned} L2-norm: \quad & f = \frac{v}{\sqrt{\|v\|_2^2+e^2}} \\ L1-sqrt: \quad & f = \sqrt{\frac{v}{\|v\|_1+e}} \end{aligned} \tag{1}$$

Nechť v je nenormalizovaný vektor obsahující všechny histogramy v daném bloku, $\|v\|_k$ je jeho k–norm pro $k = 1, 2$, a e je malá konstanta.

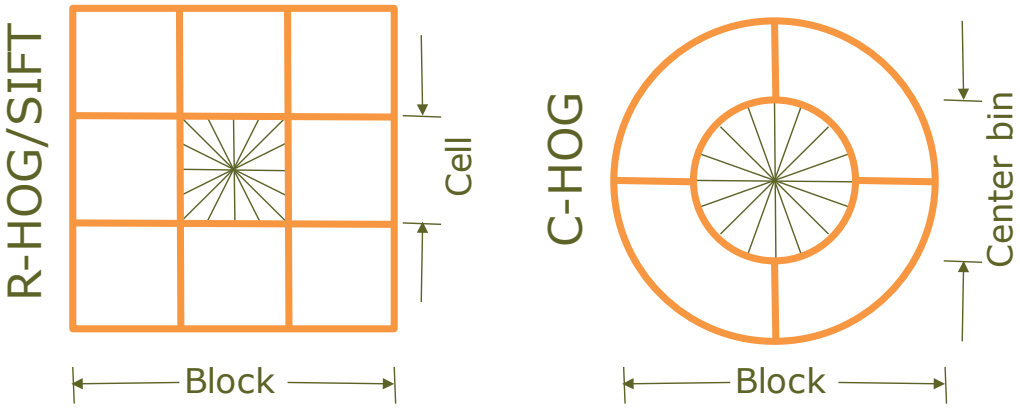


Obrázek 3: Procesní řetězec výpočtu deskriptoru [6]

C–HOG (Kruhové HOG bloky) - lze nalézt ve dvou variantách: *s jedinou, centrální buňkou* a *úhlově rozdělenou centrální buňkou*. Dají se popsat čtyřmi parametry: počtem úhlů a radiálních kanálů (binů), poloměrem centrálního binu a faktorem roztažení pro poloměr dalších radiálních

binů. Tyto bloky se podobají deskriptorům kontextu tvarů (shape context descriptors [7]), ale C–HOG jsou buňky s několika orientovanými kanály, zatímco shape context využívají přítomnost jediné hrany.

R–HOG (Obdélníkové HOG bloky) - tyto bloky jsou v praxi nejčastěji používané a reprezentují se třemi parametry: *počet buněk na blok*, *počet pixelů na buňku* a *počet binů (kanálů) na jeden histogram*. Bloky se mohou zdát podobné deskriptorům transformací příznaků invariantní vůči měřítku (SIFT, scale–invariant feature transform [8]), avšak liší se výpočtem bloků. R–HOG bloky jsou vypočteny v hustých mřížkách v libovolném měřítku bez zarovnání orientace, zatímco SIFT deskriptory obvykle v mřížkách řídkých, obrazové body invariantní vůči měřítku jsou otočeny, aby přiléhaly orientaci. R–HOG bloky se také používají pro kódování informací.



Obrázek 4: Varianty geometrie spojení bloků [6]

V knihovně OpenCV existuje funkce „detectMultiScale“, která pomocí posuvného okna (sliding window) projde celý obraz a v každém takovém okně se vypočítávají příznaky. Jeho velikost můžeme definovat pomocí parametru. Standardní velikost je 64×128 pixelů a následující popis a výpočty budou odpovídat této velikosti posuvného okna.

V tomto okně se obrázek rozdělí na 8×8 bloků a v každém bloku se vypočte histogram magnitud, které se podle směru gradientů rozdělují do 9 binů. Vyjde nám vektor o velikosti 9 a tyto vektory spojíme do bloků o velikosti 16×16 a normalizujeme je na velikost 1, aby byly nezávislé na osvětlení, dostaneme tedy vektor o velikosti 36. Na konci tohoto procesu všechny vektory spojíme a získáme vektor všech příznaků z konkrétního vzorku nebo z oblasti zájmu posuvného okna. Obrázek 6 zobrazuje vstup vzorku pro vypočítání jeho příznaků neboli histogramu orientovaných gradientů. Kde nejprve obrázek převedeme do stupňů šedi, provedeme normalizaci kontrastu a gamy a následně vypočítáme jeho vektor příznaků. Ukázka dělení obrazového okna je na obrázku 5.

Obrázek 5: Rozdělení obrazu do 8×8 buněk [9]

Vstupní obrázek Normalizace kontrastu Výpočet příznaků

Obrázek 6: Normalizace a výpočet příznaků pro obrázek

Haarovy příznaky

Na tomto přístupu je založen objektový detektor Viola–Jones (*Viola–Jones object detector framework*) [21], který poskytuje v reálném čase spolehlivou a konkurenceschopnou detekci objektů. Tento systém byl navržen v roce 2001, a i když může být vytrénován pro detekci různých objektových tříd, byl primárně použit především pro detekci obličeje. Detektor pracuje s obrazy ve stupních šedi a skládá se ze tří částí. Z integrálního obrazu, Haarových příznaků a AdaBoost[25] algoritmu. Tento algoritmus bude popsán v sekci o klasifikátorech. Integrální obraz je takový ob-

raz (obrázek 7), kde každý bod x představuje součet hodnot předchozích pixelů doleva a nahoru. Spodní pravý bod obsahuje součet všech pixelů v obraze. Zápis integrálního obrazu je:

$$I(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x',y'),$$

kde $i(x',y')$ je hodnota pixelu na pozici (x,y).

1	1	1
1	1	1
1	1	1

Vstupní obraz

1	2	3
2	4	6
3	6	9

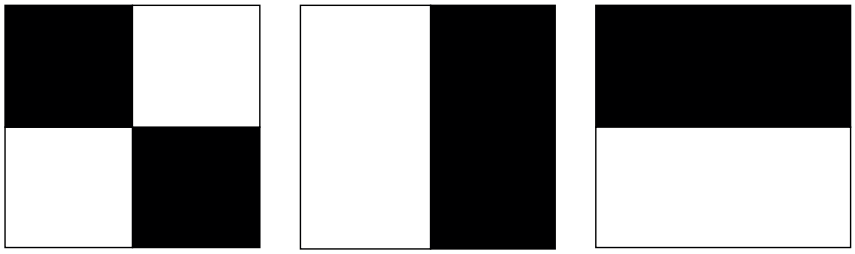
Integrální obraz

Obrázek 7: Převod obrazu na integrální obraz

Princip využití Haarových příznaků v obrazech je založen na pozorování, že lidská těla a obličeje mají některé podobné rysy. Právě tyto rysy mohou být porovnány pomocí Haarových příznaků. Jedná se například o tyto rysy:

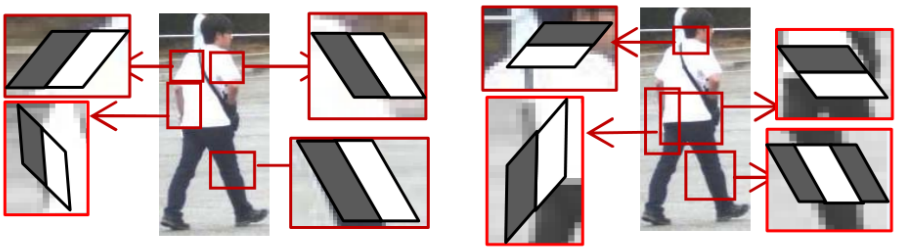
- Oční oblast je tmavší než oblast nosního mostu,
- hlava člověka je tmavší než její okolí,
- oblast mezi dolními končetinami je světlejší než samotné nohy.

Sada Haarových vlnek je na obrázku 8, jedná se pouze o základní sadu příznaků.



Obrázek 8: Základní sada Haarových příznaků

Pro identifikaci lidských postav se používá rozšířená sada vlnek, tzv. Haar–like příznaky. Autoři v publikaci [20] představili novou sadu Haar vlnek pro detekci lidí v obrazech. Klasifikační systém založený na těchto příznacích dosahuje nižší falešně pozitivní detekce než původní Haar–like příznaky. Na obrázku 9 je příklad detekce pomocí Haar–like vlnek ze zmíněné publikace. Hodnota příznaku je rozdíl mezi sumou hodnot pixelů v bílé a černé oblasti Haarových vlnek.



Obrázek 9: Použití Haar–like příznaků na chodcích [20]

Lokální binární vzor

Metoda LBP (Local Binary Pattern) byla navržena pro klasifikaci textur v obrazech v roce 1990 [22]. Poprvé však byla popsána až v roce 1994 [23]. Hlavní myšlenkou LBP je, že struktury obrazu mohou být efektivně zakódovány porovnáním hodnot jednotlivých pixelů a jejich okolí. Tato metoda je odolná vůči jasovým změnám obrazu.

Prvním krokem této metody je převod obrazu do stupňů šedi a jeho rozdělení do buněk. Okolní hodnoty pixelů jsou porovnávány se středovým pixelem, pokud je jejich hodnota rovna nebo větší zapíše se na tuto pozici jednička v opačném případě nula. Tyto hodnoty seřadíme dle hodinových ručiček nebo naopak a získáme osmimístné binární číslo, které převedeme do dekadické soustavy. V následujícím kroku z čísel, které jsme získali kombinací pixelů v buňkách, vypočítáme histogram. V posledním kroku zřetězíme všechny histogramy buněk a získáme vektor příznaků pro celý obraz. Jedná se o 256–dimenzionální vektor příznaků. Matematicky lze LPB vyjádřit jako:

$$LBP_{P,R} = \sum_{p=0} P - 1 s(g_p - g_c) 2^P, s(x) = \begin{cases} 1 & \text{pro } x \geq 0, \\ 0 & \text{pro } x < 0, \end{cases}$$

kde: P je počet bodů v okolí, R vyjadřuje vzdálenost bodů od středového pixelu, g_c je středový pixel, g_p je aktuální pixel.

Následující příklad se vztahuje k obrázku 10. Po porovnání pixelů se středovým pixelem jsme získali vzor 11110001. Tento vzor převedeme do dekadické soustavy a sečteme, $1 + 16 + 32 + 64 + 128 = 241$. Získali jsme hodnotu této buňky do vektoru příznaků.

6	2	2
7	6	1
9	8	7

Vstupní buňka

1	0	0
1		0
1	1	1

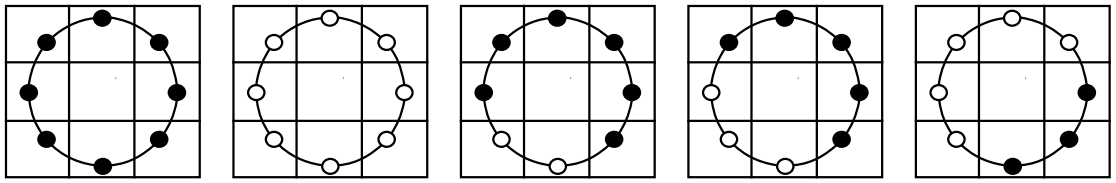
Prahové hodnoty

1	2	4
128		8
64	32	16

Pixely ohodnoceny váhou

Obrázek 10: Výpočet příznaku

Výhoda této metody je její rychlý a snadný výpočet a odolnost vůči různým osvětlením. Na druhou stranu je těžší na trénování, protože výsledné dekadické číslo může mít obrovské množství možností (podle parametru P). K omezení lze využít uniformní vzory (obrázek 11). Pro parametr $P = 8$, získáme 59 vzorů.



Obrázek 11: Lokální okolí LBP metody [24]

HOG + LBP

Kombinace techniky histogramů orientovaných gradientů a lokálního binárního vzoru tvoří nový způsob, jak vytvořit sadu příznaků. Ta je schopna zvládnout i částečnou okluzi. Jak autoři publikace [26] zmiňují, jeden detektor slouží ke globálnímu skenování obrazu a druhý detektor pro konkrétní oblasti. Oba jsou naučeny z trénovacích dat lineárního klasifikátoru SVM. Pro každé nejednoznačné skenovací okno se vytvoří pravděpodobnosti okluze dle odezvy každého bloku HOG příznaku na globální detektor (obrázek 12). Tyto pravděpodobnosti okluze se dále segmentují metodou Mean shift [28, 29]. Segmentovaná část okna s větší negativní odpovědí je označena jako okludovaná část. Pokud je indikována okluze s vysokou pravděpodobností v nějakém skenovacím okně, část detektorů je aplikována na oblasti bez okluze k dosažení konečné klasifikace aktuálního skenovacího okna.

Obrázek 12: První řádek ilustruje nejednoznačné výřezy ze skenovacího okna. Druhý řádek zobrazuje segmentové pravděpodobnostní okluze odpovídající k obrazům z prvního řádku.[26]

HOG + LUV

Tato metodika [27] využívá histogram orientovaných gradientů a LUV kanály obrazů pro extrakci příznaků. Kanály RGB vzorku se převedou na kanály LUV. Z každého kanálu se samostatně vypočtou příznaky. Tento převod ilustruje obrázek 13. Zmíněné příznaky můžeme spojit do jednoho vektoru příznaků nebo využít jen některé z nich. Kanál L je složka luminiscence (jas) a kanály UV značí sytost barev.

RGB kanály | Luminiscence (L) | Sytost barev (UV)

Obrázek 13: Rozdíl mezi RGB a LUV kanály. Obrázek je obsažen v trénovací sadě Das[33].

3.2 Detekce chodců a detekce pohybu

Další možností detekce chodců v obraze je substrakce pozadí, díky které můžeme extrahovat dynamické části v obraze. Použitím této metodiky kombinované například s přístupem založeným na příznacích, můžeme získat rychlý a optimální detektor.

Mixtura Gaussiánů

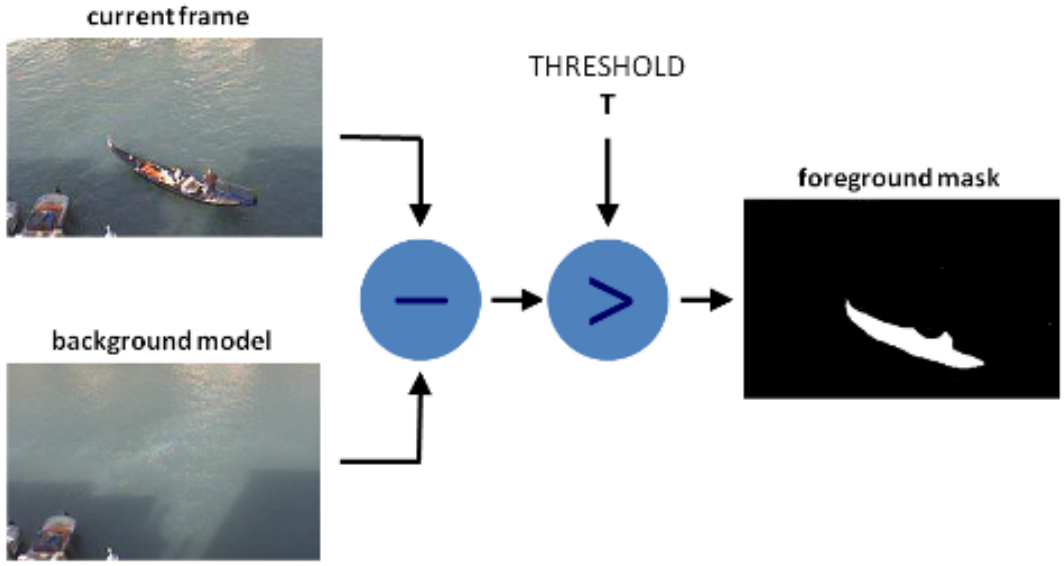
Mixtura Gaussiánů (*MOG - Gaussian mixture model background subtraction*) [4] je metoda substrakce pozadí. Je to široce používaná technika pro generování masky popředí pomocí statických kamer. Jak již lze poznat z názvu, algoritmus vypočítává masku popředí odečtením referenčního modelu od aktuálního snímku. Referenční model většinou bývá první snímek videosekvence. Dále se na masku aplikuje prahování, čímž se vyfiltruje přebytečný šum v masce. Tento postup zobrazuje obrázek 14.

Skládá se ze dvou hlavních kroků:

1. Inicializace pozadí,
2. aktualizace pozadí.

V prvním kroku je vypočítán model pozadí, zatímco v druhém kroku je tento model aktualizován, aby se přizpůsobil možným změnám ve scéně [5]. Jeho chování a citlivost detekce lze ovlivnit

nastavením parametrů. Jeho novější verze v knihovně OpenCV pak disponuje i parametrem pro detekci stínů v obraze.



Obrázek 14: Schéma metody substrakce pozadí [5]

Tento přístup, jak je výše zmíněno jen oddělí dynamické popředí od statického, proto je vhodné získat zmíněné dynamické oblasti například konvexním obalem.

Konvexní obal

Konvexní obal (*Convex hull*), je metoda sloužící k obalování pixelů v prahovém obraze. Funkce z OpenCV používá Sklanského algoritmus [2] a má složitost $O(N \log(N))$. Vstupem je množina bodů uložená v matici nebo vektoru a výstupem je vektor indexů nebo vektor bodů. Tyto body, které vytvářejí právě zmiňovaný konvexní obal se nakreslí do obrazu. Příklad konvexního obalu je na obrázku 15.

Před aplikací Po aplikaci

Obrázek 15: Aplikace konvexního obalu v obraze

3.3 Klasifikátory

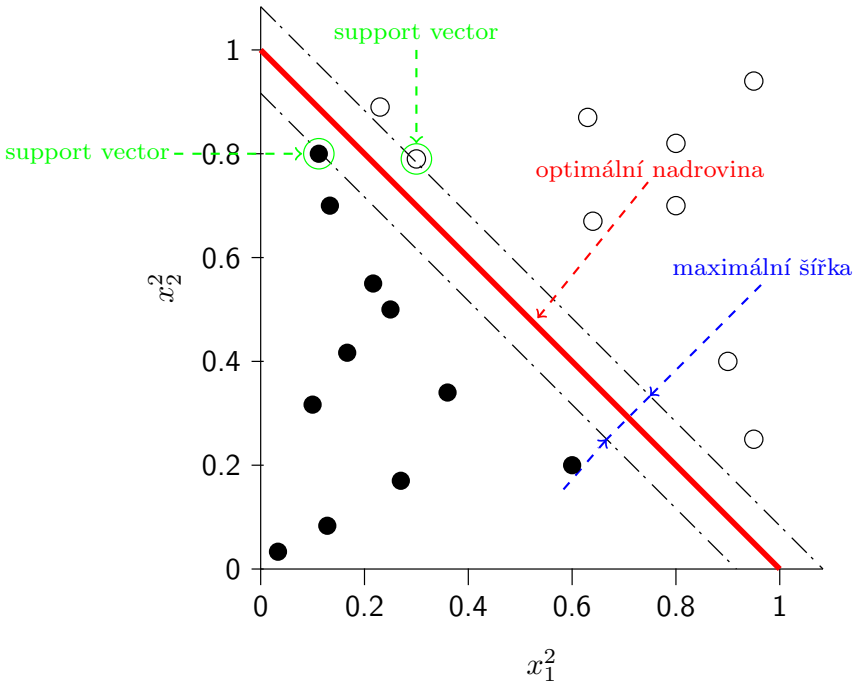
Klasifikace je obecný proces kategorizující objekty do určitých tříd. Termín klasifikátor někdy odkazuje také na matematickou funkci, implementovanou klasifikačním algoritmem.

Support vector machines

Support vector machines (SVM) jsou učební modely, které jsou velmi populární v oblasti strojového učení. Původně tato technika sloužila k vytvoření optimálního binárního klasifikátoru, později byla rozšířena na řešení problému regrese a shlukování. Tato metoda je založena na tzv. jádrových algoritmech (kernel machines) s využitím podpůrných vektorů (support vectors).

Primárním cílem SVM je nalézt nadrovinu, která optimálně rozděluje prostor příznaků tak, aby trénovací data náležela do konkrétních tříd. Tuto nadrovinu ilustruje obrázek 16. Pokud mezera mezi oddělující nadrovinou a nejbližšími vektory příznaků z obou kategorií (v případě binárního klasifikátoru) je maximální, jedná se o optimální řešení. Vektory příznaků v blízkosti této nadroviny se nazývají podpůrné vektory, což znamená, že pozice ostatních vektorů nemá vliv na nadrovinu (rozhodovací funkce).

Jinými slovy, se jedná o diskriminační klasifikátor formálně definovaný rozdělovací nadrovinou, která kategorizuje nové příklady.



Obrázek 16: Optimální oddělovací hranice

Implementaci SVM lze nalézt v již existujících knihovnách, jako jsou například LIBSVM [10], kernlab [11], scikit-learn [12], SVMLight [13] a další. V následující části bude popsána knihovna LIBSVM, na které je založena implementace v knihovnách OpenCV a v Dlib.

Tato knihovna byla napsána v roce 2001 [10] a stále se vyvíjí. Podporuje různé formulace SVM pro odhady klasifikace, regrese a distribuce.

C–Support vektorová klasifikace (C–SVC) Umožnuje nedokonalé oddělení tříd pro n–tříd ($n > 2$) s postihovým multiplikátorem C, pro odlehlé hodnoty ($C > 0$) [16].

ν–Support vektorová klasifikace (ν–SVC) n–třídní klasifikace s možností nedokonalé separace. Tato klasifikace přidává nový parametr $\nu \in (0; 1\rangle$, čím větší je jeho hodnota, tím hladší je rozhodovací funkce [17].

Distribuční odhad (Jednotřídní SVM) Distribution Estimation (One-class SVM), jak již název sám o sobě napovídá všechny trénovací data pocházejí z jedné třídy, SVM vytvoří hranici, která odděluje třídu od zbývající části [18].

ε–Support vektorová regrese (ε–SVR) Vzdálenost mezi vektory příznaků a rozdělovací nadrovinou musí být menší než mez tolerance ε. Pro odlehlé hodnoty opět použijeme multiplikátor C. Musí tedy platit: $C > 0$ a $\varepsilon > 0$ [19].

ν–Support vektorová regrese (ν–SVR) Tato klasifikace je podobná jako ε–SVR. Na místo ε se použije parametr $\nu \in (0; 1\rangle$ [17].

Účinnost SVM závisí na výběru správného jádra a jeho parametrů. Často se používá Gaussovo jádro s jedním parametrem γ. Díky jeho přesnosti, ale je časově náročné. V této knihovně se můžeme setkat s následujícími jádry.

Lineární jádro Použití tohoto jádra je velmi rychlé (bez jakékoliv transformace), jedná se o lineární diskriminaci a rozdělovací nadrovina bude vždy přímka. Pro toto jádro platí

$$K(x_i, x_j) = x_i^T x_j,$$

kde x_i a x_j jsou vektory vstupního prostoru.

Polynomické jádro Polynomické jádro umožňuje učení nelineárních modelů

$$K(x_i, x_j) = (\gamma x_i^T x_j + c)^d, \gamma > 0,$$

kde: $c \geq 0$, volný parametr, který vylučuje vliv vyššího řádu oproti polynomu nižšího řádu (pokud $c = 0$, jádro je homogenní), řád polynomu určuje parametr d.

Gaussovo jádro Gaussovo neboli RBF (Radial Basis Function) jádro se řadí mezi nejpoužívanější a je definované jako

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0,$$

kde: $||x_i - x_j||^2$ značí kvadratickou euklidovskou vzdálenost mezi dvěma vektory příznaků.

Sigmoidní jádro toto jádro je podobné sigmoidní funkci v logistické regresi

$$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r),$$

kde r je volitelný parametr.

Exponenciální jádro Exponenciální jádro $\chi 2$ je podobné RBF jádru a využívá se převážně na histogramy

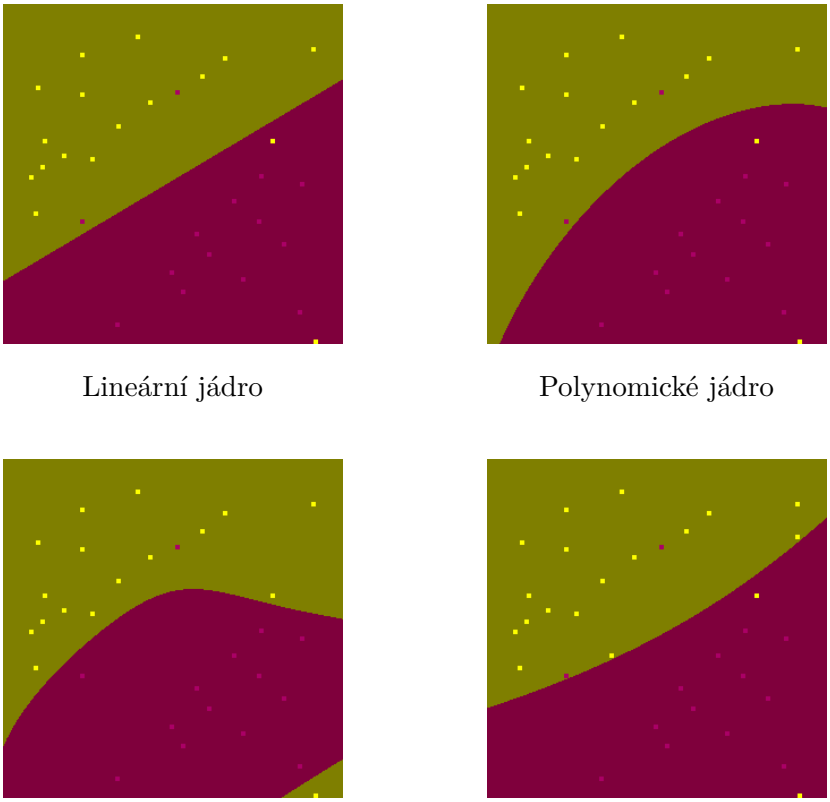
$$K(x_i, x_j) = e^{-\gamma\chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = \frac{(x_i - x_j)^2}{(x_i + x_j)}, \gamma > 0,$$

Jádro histogramu průsečíků Toto jádro je také známé jako *Min Kernel*, jedná se o nejnovější jádro v této knihovně a je velmi rychlé a užitečné při klasifikaci

$$K(x_i, x_j) = min(x_i, x_j).$$

Nejpoužívanější jádra jsou ilustrována na obrázku 17.

SVM je výsledek práce několika lidí po mnoho let. První algoritmus této problematiky je přisuzovaný Vladimíru Vapnikovi v roce 1963 [14]. V reálném životě byly úspěšně použity ve třech hlavních oblastech: kategorizace textu, rozpoznání obrazu a bioinformatika. Mezi konkrétní příklady patří třídění novinových zpráv, rozpoznávání ručně psaných čísel nebo například vzorky rakovinových tkání.

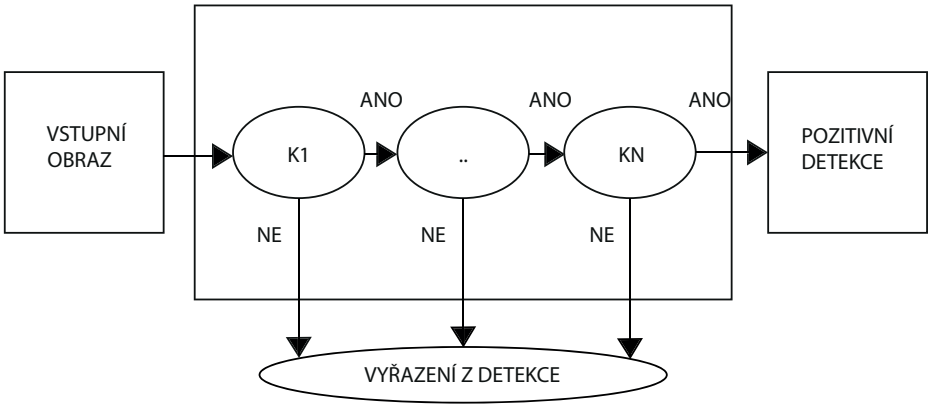


Obrázek 17: Ilustrační obrázek jader při použití C–SVM [10]

Kaskádové klasifikátory

Kaskádový klasifikátor se skládá z více slabších klasifikátorů umístěných v kaskádách za sebou. Požadavky na tento druh klasifikátoru byly rychlost detekce, aby mohl být implementován na procesorech s nižším výkonem, například v kamerách nebo v telefonech. Princip tohoto klasifikátoru je velice jednoduchý. Klasifikátor na první vrstvě může vyfiltrovat většinu negativních oken. Na druhé vrstvě se mohou odfiltrovat „těžší“ negativní okna, která přežila z první vrstvy a tak dále. Subokno, které přežije všechny vrstvy, bude označeno jako pozitivní detekce. Příklad řetězce kaskádového klasifikátoru je ilustrován na obrázku 18, kde $K1$–KN je klasifikátor první až n–té vrstvy.

Klasifikátory si mezi sebou předávají všechny informace o vstupním obraze. Tímto kaskádovým vyhodnocováním se může redukovat čas, nutný pro detekci v daném obraze. Prvním takovým klasifikátorem byl detektor obličeje Viola–Jones [21].



Obrázek 18: Ukázka pipeline kaskádového klasifikátoru

AdaBoost

AdaBoost, neboli Adaptive Boosting, byl představen v práci od Freund a Schapire [25]. Tento klasifikátor kombinuje slabé klasifikátory k vytvoření jednoho silného klasifikátoru. V kombinaci více klasifikátorů s výběrem trénovací sady v každé iteraci algoritmu a přidělení správné váhy na konci trénování, docílíme klasifikátoru s dobrou přesností. Klasifikátory v tomto řetězci, které mají klasifikační přesnost menší než 50%, jsou ohodnoceny zápornou vahou. Váhou nula jsou ohodnoceny klasifikátory, které mají přesnost 50%. Pouze ty, keré mají přesnost vyšší než 50%, jsou přínosné do této kombinace a můžeme hovořit o zesílení (boosting) klasifikace.

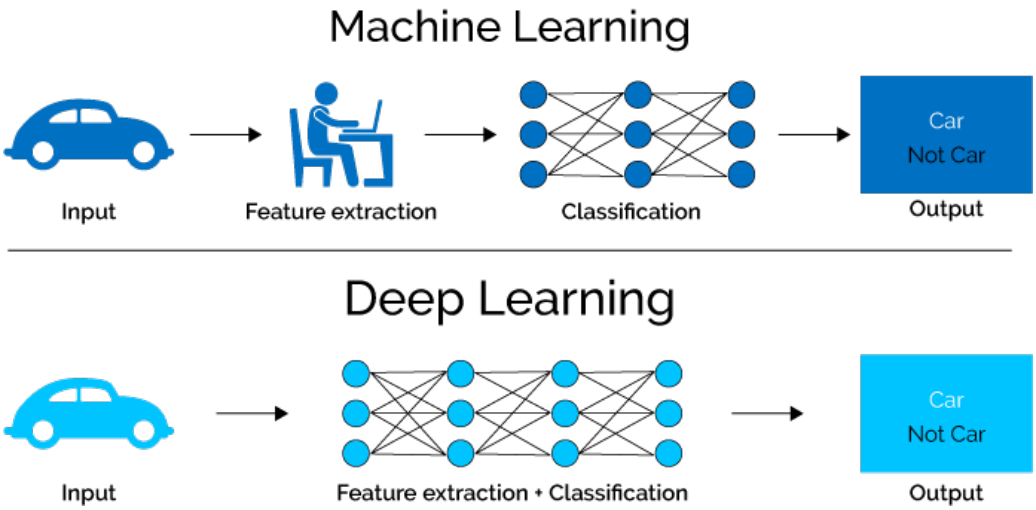
3.4 Deep learning

Deep learning neboli hluboké učení, známé také jako hierarchické učení, je sbírka algoritmů používaných ve strojovém učení. Používají se k modelování abstrakcí na vysoké úrovni v datech za pomocí modelových architektur, které se skládají z několika nelineárních transformací. Hluboké učení je součástí široké skupiny metod používané pro strojové učení, které jsou založeny na učení reprezentace dat. Hluboké strukturované učení může být:

- **Kontrolované (s učitelem)** - všechna data jsou kategorizovaná do tříd, algoritmy se učí předpovídat výstup ze vstupních dat.
- **Částečně kontrolované** - data jsou částečně kategorizovaná do tříd. Pří tomto přístupu učení lze využít kombinaci kontrolovaného a nekontrolovaného přístupu učení.
- **Nekontrolované (bez učitele)** - data nejsou kategorizovaná do tříd, algoritmy se učí ze struktury vstupních dat.

Hluboké učení je specifický přístup, použitý k budování a učení neuronových sítí, které jsou považovány za velmi spolehlivé rozhodovací uzly. Jestliže vstupní data algoritmu procházejí řadou nelinearit a nelineárních transformací, tak tento algoritmus je považován za „deep“ algoritmus.

Odstraňuje také ruční identifikaci příznaků (obrázek 19) z dat a místo toho se spoléhá na jakýkoliv trénovací proces, které má za úkol zjistit užitečné vzory ve vstupních příkladech. To dělá neuronovou síť jednodušší a rychlejší, a může přinést lepší výsledky než z oblasti umělé inteligence.

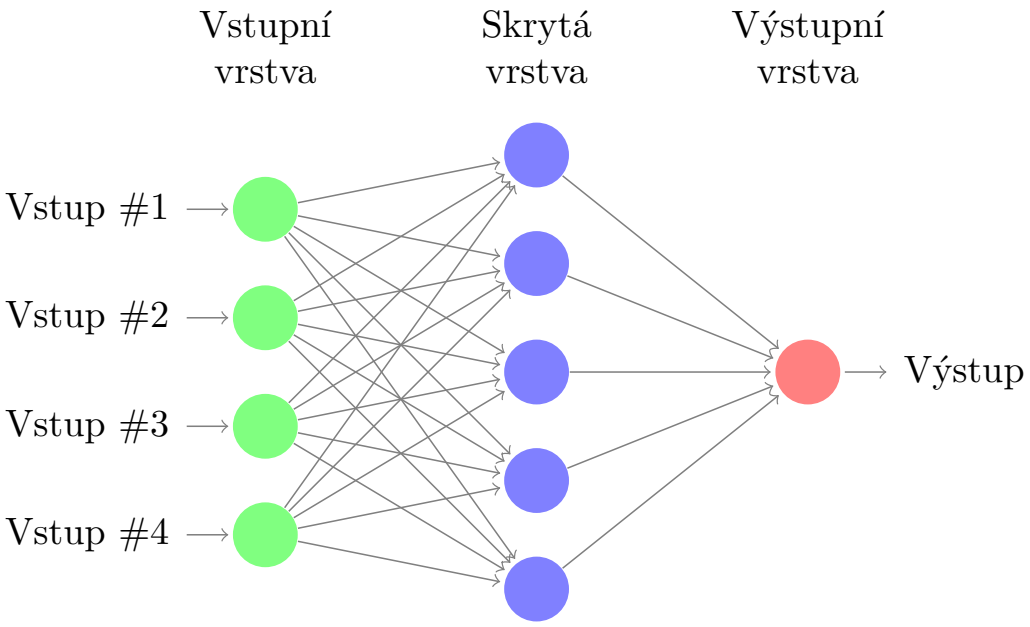


Obrázek 19: Hlavním rozdílem mezi strojovým a hlubokým učením je ten, že u strojového se příznaky musí extrahovat manuálně. [30]

Neuronové sítě

Neuronové sítě (*ANN* - Artificial Neural Network) jsou inspirované lidským mozkem [31], který je složený z různých vzájemně propojených vrstev neuronů, kde každý z nich přijímá informaci z předchozího, zpracovává tuto informaci a odesílá jí do dalšího neuronu, dokud není přijat konečný výstup. Může se jednat o výstup s danou kategorií, jestliže se jedná o kontrolované učení nebo o určitá kritéria v případě nekontrolovaného učení.

Výhodou ANN nad ostatními strojovými učícími strategiemi, jako je například SVM je to, že ANN je pravděpodobnostní klasifikátor umožňující klasifikaci více tříd. To znamená, že může zajistit více než jeden objekt v obraze, na rozdíl od SVM klasifikátoru, který je pravděpodobnostní binární klasifikátor. Příklad topologie neuronové sítě je na obrázku 20.



Obrázek 20: Neuronová síť je propojená skupinou uzlů, podobná síti neuronů v mozku.

Typickým příkladem neuronové sítě je vícevrstvý perceptron (ANN–MLP). Tato neuronová síť se skládá minimálně ze tří vrstev uzlů (vstupní, výstupní a skrytou). Každý z uzlů je neuron, který využívá nelineární aktivační funkci s výjimkou vstupních uzlů:

- **Vstupní vrstva** - jedná se o pasivní vrstvu, která nemodifikuje data, pouze je získává z okolního světa a pošle je dál do sítě. Počet uzlů v této vrstvě závisí na množství příznaků nebo deskriptivních informací, které chceme extrahovat z obrázku.
- **Skrytá vrstva** - v této vrstvě probíhá transformace vstupů do něčeho, co může výstupní nebo jiná skrytá vrstva využít (za předpokladu, že existuje více skrytých vrstev). Počet uzlů je určen složitostí problému a přesnosti, které chceme přidat do sítě.
- **Výstupní vrstva** - tato vrstva musí také vždy existovat v topologii sítě, ovšem počet uzlů v tomto případě bude definován vybranou neuronovou sítí. Pokud detekujeme na obrázku pouze jeden objekt, bude mít vrstva jen jeden uzel (lineární regrese) a bude vracet hodnotu definující pravděpodobnost konkrétního objektu v rozmezí $[-1, 1]$.

Vysoká dimenze vstupního vektoru zvyšuje přesnost výsledků, ovšem na druhou stranu zvyšuje výpočetní náklady. Dalším rozhodnutím je použití aktivační funkce pro skrytou vrstvu, která umožnuje přizpůsobit nelineární hypotézy a získat lepší detekci vzoru v závislosti na poskytnutých datech. Běžná volba aktivační funkce pro vícevrstvý perceptron je Sigmoid, ale může být použita funkce tanh nebo ReLU. Při hlubším pohledu na skrytou vrstvu lze říct, že všechny neurony se chovají podobně. Hodnoty jsou získávány z předchozí vrstvy, sečteny s určitými váhami a hodnotou zkreslení. Suma těchto hodnot je transformována pomocí aktivační funkce, která se může také lišit pro různé neurony.

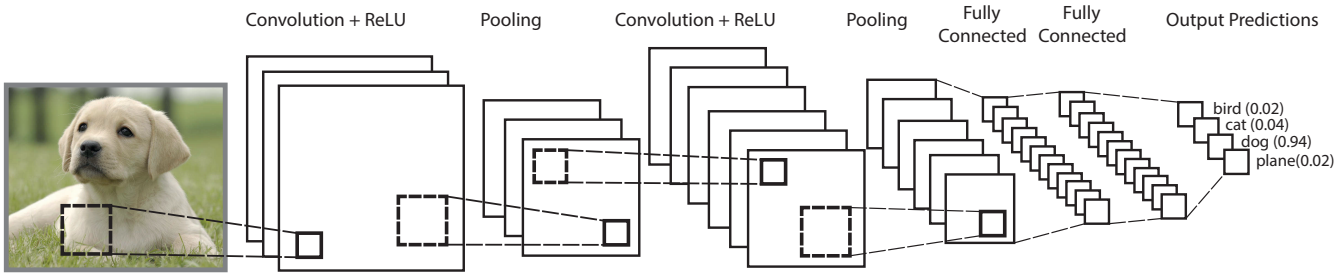
Konvoluční neuronové sítě

Další možností je použití konvoluční neuronové sítě (*CNN* - Convolution neural network) [32]. Tyto sítě jsou speciálním druhem vícevrstvých neuronových sítí a jsou navrženy tak, aby rozpoznaly vizuální vzory přímo z pixelu obrazu s minimálním předzpracováním. Mohou rozpoznat vzory s extrémní variabilitou (například ručně psané znaky) a odolnost vůči deformacím a jednoduchým geometrickým transformacím. Název „Konvoluční neuronové sítě“ naznačuje, že síť využívá matematickou operaci zvanou konvoluce alespoň v jedné jejich vrstvě.

Nejznámější a nejvíce používanou konvoluční neuronovou sítí jsou modely LeNet [32]. Hlavní kroky LeNet sítě jsou:

- **Konvoluce** - tyto vrstvy provádějí konvoluci nad vstupy do neuronové sítě.
- **Nelinearita (ReLU)** - tato vrstva je použita po každé konvoluční vrstvě a jejím cílem je nahrazení všech negativních pixelů nulou ve výstupu této vrstvy (příznaková mapa).
- **Pooling/sub sampling** - ze vstupního obrazu vyextrahuje pouze zajímavé části pomocí některých matematických operací (max, avg, sum), a tím se redukuje jeho dimenzionalita.
- **Fully connected layer/klasifikace** - tato vrstva vychází z původních umělých neuronových sítí, konkrétně z vícevrstvého perceptronu. Tato vrstva je typicky umístěna na konci sítě a je propojena s klasifikační vrstvou pro predikci.

Tyto vrstvy jsou ilustrované na obrázku 21.



Obrázek 21: Řetězec LeNet konvoluční neuronové sítě [32]

4 Datasety chodců

V průběhu let bylo shromážděno nespočet trénovacích sad pro chodce, které jsou veřejně dostupné na internetu. Všechny mají rozdílné charakteristiky, slabé a silné stránky.

Sada INRIA [34] patří mezi nejstarší sady. Ačkoliv obsahuje poměrně málo vzorků, přináší velmi kvalitní anotace chodců v různých prostředích, což je většinou také hlavní důvod, proč je tato sada vybrána pro trénování. Na rozdíl sady ETH [35] a TUD-Brussels [36] patří mezi středně velké video sady. Další známou trénovací sadou je Daimler–Mono [38], kterou nelze použít pro všechny metodiky, poněvadž poskytuje vzorky v odstínech šedi. Sady ETH, KITTI [37] a Daimler–Stereo [39] poskytují stereofonní informace. Na obrázku 22 je ukázka trénovací a testovací sady Daimler–Stereo.

Obrázek 22: Ukázka trénovací a testovací sady Daimler–Stereo [39]

V dnešní době převládají Caltech-USA [40] a KITTI jako trénovací sady pro chodce. Obě sady poskytují poměrně velký počet vzorků. Caltech-USA vyniká počtem možných využití a obsahuje více než 2300 jedinečných anotací chodců. KITTI zase předčí svou testovací sadou, která je více rozmanitější, ale na druhou stranu se tato sada nepoužívá tak často. V roce 2014 vznikl projekt PETA (Pedestrian Attribute dataset)[41]. Tento projekt kombinuje mnoho trénovacích sad a vytváří tak jednu velkou a rozmanitou sadu, která usnadňuje učení robustních detektorů s dobrým výkonem.

V této práci používám převážně trénovací sadu CUHK01 [42], sadu Das [33] a jako negativní sadu Daimler–Mono [38]. Tato negativní sada se vyskytuje v podobě fotografií zachycených kamerou při jízdě autem, bylo jí tedy nutné před použitím zpracovat.

5 Aplikace

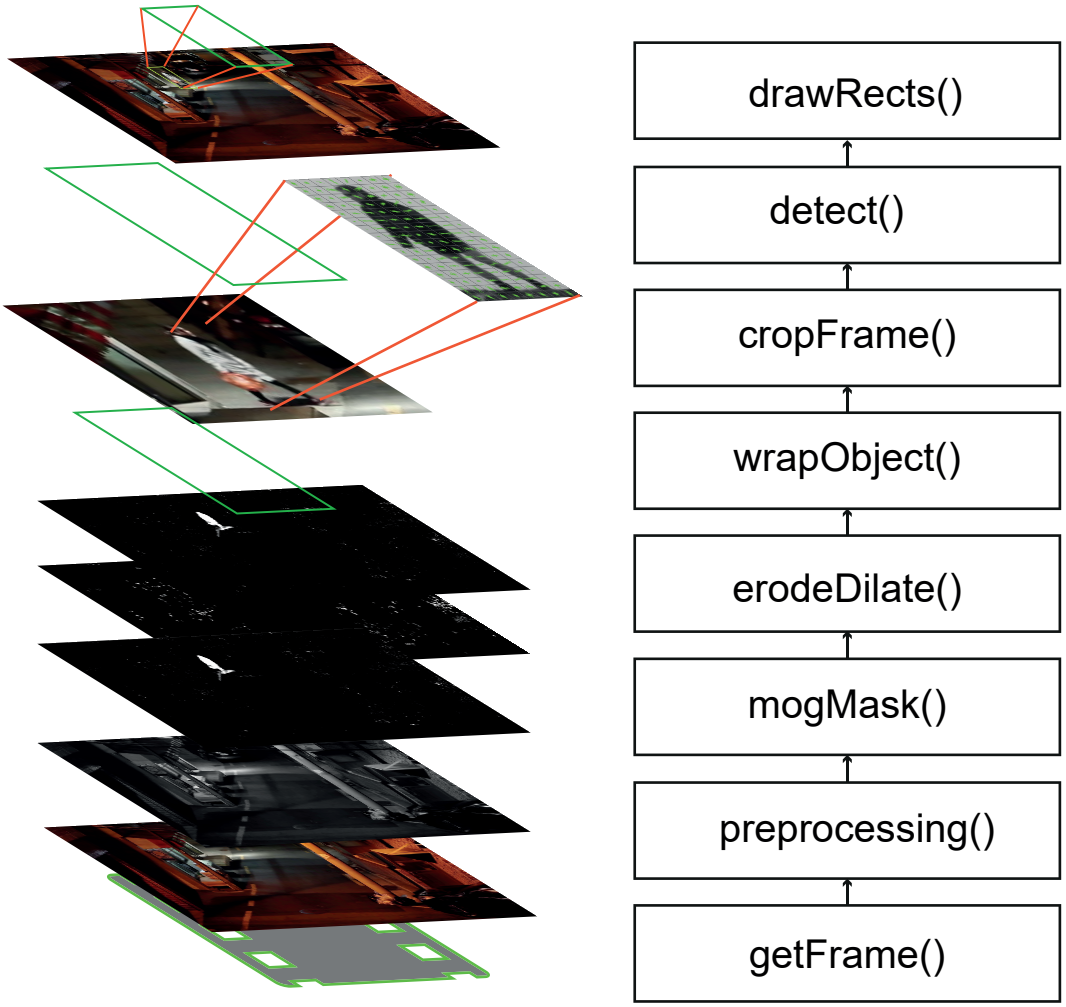
Aplikace je napsaná v nativním C++–11, ve vývojovém prostředí Visual Studio 2015. Jedná se o multiplatformní konzolovou aplikaci, která byla primárně určena pro systémy ARM s operačním systémem Linux. Hlavním účelem této aplikace je detekování chodců v obrazech.

Její obsluha je na bázi argumentů, které určují její chování. Nastavení parametrů se provádí přes externí soubor, což umožňuje jednodušší a pohodlnější manipulaci s aplikací. Disponuje širokým spektrem nástrojů od testování klasifikátorů, po jejich trénování až po samotnou detekci. Její nástroje jsou popsány v následujících podsekcích. Poslední podsekce je věnována použitým knihovnám práce.

5.1 Detekování chodců

Aplikace umožnuje detekování chodců ve videosekvencích, obrázcích a také z webové kamery. Díky externímu nastavení prahů detektorů a jiných proměnných lze měnit nastavení bez opětovného kompilování aplikace. Součástí aplikace je anotační nástroj, který slouží k označení oblasti s chodci. Nástroj zaznamená výstup do souboru, kterým se vyhodnotí výstup detektoru.

Obrázek 23 ilustruje průběh algoritmu při volbě detekce pomocí metody MOG a HOG. Prvním krokem algoritmu je získání snímku z aktuální videosekvence nebo kamery (getFrame), pokud obrázek je prázdný, cyklus zde končí. Snímek se následně předzpracovává převodem do stupňů šedi a aplikací filtru rozostření (preprocessing). Tento obrázek je následně zpracováván metodikou substrakce pozadí (mogMask) a na tento obraz se aplikuje eroze a dilatace(erodeDilate), proto, aby se v snímku vyfiltrovaly malé, nevýznamné oblasti a následně spojily ty větší potencionální, kde se může nacházet chodec. Tyto oblasti se obalí pomocí kontur a převedou se na obdélníky (wrapObject), díky kterým se z obrazu vystřihnou tyto části (cropFrame). Na získaných oblastech v následujícím kroku probíhá samotná detekce (detect). Detekční metoda vrátí obdélníkové oblasti potencionálních chodců a vykreslí je do původního obrazu (drawRects). Proces se opakuje do té doby, dokud získaný obrázek z kamery není prázdný. Pokud obrázek po aplikování masky substrakce pozadí neobsahuje dostatečně velké oblasti k aplikaci rámců, program pokračuje načítáním dalšího obrazce z kamery.



Obrázek 23: Algoritmus aplikace při selekci kombinace MOG a HOG detekce

5.2 Křížová validace klasifikátoru

Křížová validace (cross validation) je metoda zjišťování, jak moc bude daný klasifikátor spolehlivý v pozitivní detekci. Vytrénovaný klasifikátor se otestuje na sadě vzorků, které nebyly součástí jeho tréninku a známe jejich přesnou kategorii zařazení.

Aplikace nabízí křížovou validaci OpenCV a Dlib klasifikátoru. Diskriminační klasifikátor SVM z OpenCV lze validovat třemi způsoby. První je náhodný přístup, což je generování trénovacích parametrů klasifikátoru s jejich různou iterací.

Dalším, efektivnějším způsobem je validace pomocí diferenciální evoluce. Tento algoritmus vychází z genetického žíhání. Algoritmus nastaví určitý počet dimenzí pro validaci parametrů SVM a následně probíhá „žíhání“ parametrů takřka k jejich dokonalosti. Tento proces trvá 1000 cyklů.

Posledním druhem validace pro klasifikátor SVM z OpenCV je podobný z Dlib knihovny, jedná se o vnořené cykly, kde se hrubou silou postupně testují parametry a výsledek validace se vypisuje do konzole.

Pro klasifikátor z Dlib knihovny je použita pouze metodika pro postupné testování parametrů hrubou silou v cyklech, která je dostupná v knihovně. Výhodou této validace je rozdělení trénovacích dat do tří množin, kde dvě slouží na trénování, jedna pro testování a následně se prohodí. Tímto zajistíme větší přesnost trénovacích parametrů. Zmíněná validace má nejdelší výpočetní čas a jejím výsledkem je přesnost klasifikátoru pro pozitivní a negativní množinu vzorků.

5.3 Testování klasifikátoru

Testování klasifikátoru je proces, při kterém se otestuje na daných trénovacích vzorcích, u kterých známe jejich třídu. Díky této metodice zjistíme, jak spolehlivě je vytrénovaná SVM a jaká je její přesnost detekce. Přesněji řečeno, k daným vzorkům máme soubor „ground truth“ hodnot, proti kterým se porovnává výstup predikce daného klasifikátoru.

5.4 Trénování klasifikátoru

Program také umožňuje vytrénování vlastního klasifikátoru. Parametry se vyberou z externího souboru s nastavením. V tomto souboru jsou také uloženy cesty k souborům se vzorky.

Trénovací režim aplikace lze spustit přes argument '-t=train"a následně je uživateli nabídnut typ trénování. Program zvládne vytrénovat klasifikátor jak z knihovny OpenCV, tak i z knihovny Dlib.

Trénování klasifikátorů z OpenCV probíhá naplněním vzorků do paměti včetně jejich kategorizace. Následně jsou pro každý vzorek vypočítané jeho příznaky a opět uloženy do paměti programu. Posledním krokem před trénováním je jejich převod do jednořádkové matice a poté je spuštěno samotné trénování klasifikátorů. Tento krok trvá v závislosti na velikosti trénovací matice a zvolených parametrů. Uživatel může zvolit i dvojité trénování, což je stejný postup, avšak na konci trénování se spustí detekce pomocí posuvného okna na negativních vzorcích a tyto vzorky budou rozšířeny o výstup z detektoru. Tento proces se nazývá „Bootstrapping“ [45]. Výstupem trénování je soubor YAML, který je strojově čitelný a slouží pro serializaci strukturovaných dat. Nalezneme zde parametry, použité pro trénování, vytrénované vzorky a vektor, který určuje rozdělující přímku.

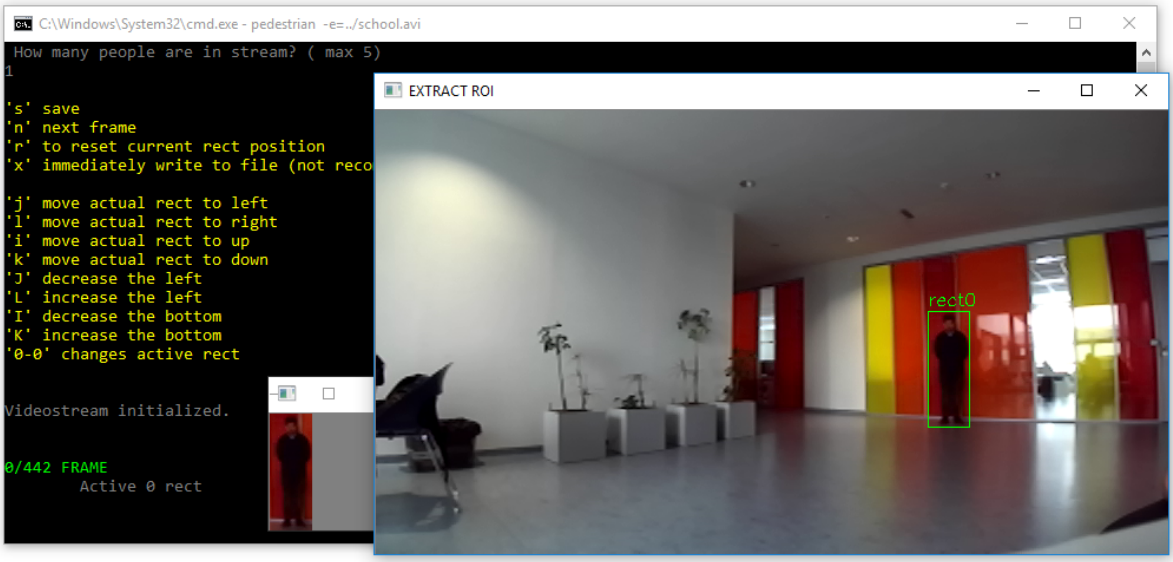
Další možností je vytrénování klasifikátoru z knihovny Dlib. Trénování je obdobné jako u předchozí metody.

Poslední možností trénování je kombinace výše zmíněných postupů. Vzorky se zpracují za pomocí OpenCV knihovny a vstupem do trénovací metody klasifikátoru z knihovny Dlib je trénovací matice. Matici je nejprve nutné převést na formát této knihovny, a poté je možné provést samotné trénování. Výstupem trénovací metody z knihovny Dlib je binární soubor.

Trénování klasifikátoru je velmi náročná operace na hardwarové prostředky. Ovšem závisí na trénovací sadě. Sada o počtu 200 tisíc vzorků zkonzumuje až 16 GB paměti.

5.5 Anotace chodců v obraze

Do výsledného programu jsem naimplementoval nástroj pro anotaci chodců v obraze. Pomocí tohoto nástroje můžeme anotovat až 5 chodců (každá anotace má svou barvu). Při vývoji jsem se snažil o to, aby manipulace s tímto nástrojem byla jednoduchá a intuitivní. Náhled nástroje je na obrázku 24. Anotovaná část se může překrývat s jinou. Konkrétní anotace je zobrazena pro kontrolu samostatně zobrazena v dalším okně. Všechny tyto objekty zájmu a snímek s anotacemi jsou uloženy na disk v podobě obrázku pro pozdější kontrolu nebo objekty zájmu mohou být použity jako pozitivní vzorky pro trénování nebo křížovou validaci klasifikátoru. Výstup tohoto nástroje je plně kompatibilní s evaluační funkcí programu a jedná se tedy o „ground truth“ soubor všech anotací objektů zájmu ve videosnímku. Stručný manuál k nástroji je popsán v příloze B.



Obrázek 24: Nástroj pro anotování oblastí chodců

5.6 Tvorba negativních vzorků z obrazu

Tento nástroj umožňuje vytvořit negativní vzorky pro trénování procházením obrázku pomocí posuvného okna. Velikost posuvného okna je definována v externím souboru. Vstupem je textový soubor s cestami k obrázkům.

5.7 Použité knihovny

Na internetu je spousta dostupných či placených knihoven pro zpracování obrazu. Avšak nejznámější a nejpoužívanější knihovnou pro zpracování obrazu je OpenCV [43]. Tato knihovna je dostupná na většinu platforem a z tohoto důvodu jsem jí v práci použil. Dále jsem v práci použil knihovnu Dlib [44], která je taky volně dostupná na internetu a také se jedná o multiplatformní knihovnu. Obě tyto knihovny budou popsány v následujících dvou podsekcí.

5.7.1 Knihovna OpenCV

OpenCV (*Open Source Computer Vision Library*) je open-source knihovna, která slouží ke zpracování obrazu a strojovému učení. Je psaná v C/C++ jazyce s podporou multijádrového zpracování.

Knihovna má více než 2500 optimalizovaných algoritmů, které zahrnují obsáhlé sady klasických a moderních algoritmů pro zpracování obrazu a strojové učení. Tyto algoritmy mohou být použity na detekci a rozpoznávání tváří, identifikování objektů, vyhodnocování lidských akcí ve videosekvencích, sledování pohybů pomocí kamery, sledování pohybujících se objektů, extrahování 3D modelů z objektů, vytváření 3D mračen bodů ze stereo kamer, spojování obrazu do jednoho s vysokým rozlišením na dané scéně, hledání podobných obrazů z obrazové databáze, odstranění červených očí z obrazu způsobené bleskem, sledování pohybu očí, rozeznání scenérie a vytvoření značek pro překrytí rozšířenou realitou a další.

Knihovna má rozhraní pro jazyky C++, C, Python, Java a MATLAB a podporuje operační systémy Windows, Linux, Android, iOS a MacOS. Pokud jsou dostupné MMX a SSE instrukce tak knihovna je použije pro zvýšení jejího výkonu.

Podílí se na ní komunita lidí, kterou tvoří více než 47 tisíc uživatelů z celého světa a přesahuje více než 14 milionů stažení.

Z této knihovny byly použité následující nástroje:

- Mixtura Gausiánů,
- Konvexní obal,
- Histogram orientovaných gradientů,
- Support vector machines.

5.7.2 Knihovna Dlib

Autorem této knihovny je Davis King. Jedná se o moderní C++ knihovnu obsahující algoritmy pro strojové učení a nástroje pro vytváření komplexních programů v jazyce C++. Používá se jak v industriální, tak v akademické sféře v široké škále oblastí, jako jsou zejména vestavěná zařízení, robotika, mobilní telefony a velká, výkonná výpočetní prostředí. Jedná se o sbírku nezávislých

softwarových komponent, kde každá z nich je doprovázena důkladnou dokumentací a mnoha příklady použití.

Jádrem filozofie této knihovny je věnování se snadnému používání a přenositelnosti. Proto je kód navržen tak, aby nebylo po uživateli vyžadováno cokoli ručně konfigurovat nebo instalovat. K dosažení tohoto cíle je veškerý kód specifický a pro konkrétní platformu omezený a obalený pomocí API rozhraní. Všechno ostatní je buď navrstveno na těchto obalech nebo napsáno v normě ISO standardu C++.

Knihovna se stále rozrůstá hlavně díky dobrovolným přispěvovatelům a v době psaní práce obsahuje například i softwarové komponenty pro práci se sítí, vlákna, grafické rozhraní, komplexní datové struktury, lineární algebru, statistické strojové učení, zpracování obrazu, data mining, XML a parsování textu, numerickou optimalizaci, Bayeské sítě. V uplynulých letech byla velká část vývoje zaměřena na širokou sadu nástrojů statického strojového učení, avšak knihovna zůstává univerzální.

V současné době je známo, že knihovna pracuje na systémech OS X, MS Windows, Linux, Solaris, BSD, HP-UX. Knihovna by měla také pracovat na libovolné platformě POSIX, ale není otestovaná na všech dostupných verzích. Z této knihovny byl v práci použit pouze FHOG detektor objektů.

6 ARM zařízení

ARM zařízení, respektive embedded systémy, jsou kompletní počítače vybudované na jedné desce s procesorem, pamětí, vstupy, výstupy a dalšími funkcemi. Jednodeskové počítače (*SBC* - single-board computer) byly vyrobeny za účelem vývoje aplikací nebo jejich demonstraci. Slouží také jako systémy pro vzdělávání nebo jsou použity jako vestavěné počítačové kontroléry. Typickým využitím těchto malých počítačů pro domácnost je její automatizace. Mezi známé a spíše výkonnější, můžeme zařadit následující vestavěné systémy.

Nvidia JETSON TK1

Tento embedded systém disponuje nejen čtyřjádrovým procesorem Cortex-A15, ale i grafickým čipem Nvidia Kepler s 192 Cuda jádry. Tato kombinace je jinak nazvaná jako Tegra K1 SoC (System on a chip – integrovaný v jediném obvodě). Tento procesor je údajně o 50 % výkonnější než Cortex-A9[1] a podporuje instrukční sadu Armv7-A.

Asus Tinker Board

Asus Tinker Board se řadí podle specifikace mezi výkonnější systémy. Na této desce se nachází čtyřjádrový procesor Cortex-A17 s možností dynamického přetaktování (Turbo-Boost) až na 2,6 GHz. Tento počítač také podporuje instrukční sadu Armv7-A.

Rock64

Rock64 je produktem firmy Pine64 a je osazen čtyřjádrovým procesorem Cortex-A53 a až 4 GB operační pamětí. Dále vyniká s konektivitou USB 3.0 a 128 MB sériovou flash pamětí. Procesor podporuje instrukční sadu Armv8-A.

Odroid-XU4

Tento počítač je vybaven osmijádrovým mobilním procesorem Samsung Exynos 5422. Tento procesor nabízí čtyřjádrový Cortex-A15 s taktem 2,1 GHz a další čtyřjádrový Cortex-A7 s taktem 1,4 GHz, což dělá tento počítač velmi výkonným na paralelní práci. Tento procesor také podporuje instrukční sadu Armv7-A.

Intel Galileo

Intel Galileo je prvním nízkoodběrovým produktem této firmy. Disponuje jednojádrovým procesorem Intel Quark SoC X1000 s taktem 400 MHz. První generace byla dostupná již v roce 2013. Deska je navržena pro IoT (*Internet of Things*) a je zcela kompatibilní s produkty, knihovnami a vývojovým prostředím elektrické platformy Arduino.

[1] https://developer.arm.com/products/processors/cortex-a/cortex-a15

Orange PI Plus 2E

Orange PI je počítač velmi podobný Raspberry PI, ovšem svým výkonem je spíše obdobný počítači Banana PI. Orange PI Plus 2E je vybaven čtyřjádrovým procesorem Cortex-A7 s podporou instrukční sady Armv7-A, 2 GB operační pamětí nebo například WiFi modulem s anténou. Výhodou tohoto počítače je využití standardního DC konektoru k napájení.

Cubieboard 5

Tento počítač je osazen SoC čipem Allwinner H8. Jedná se o symetrické osmijádro složené z Cortex-A7 o frekvenci 2,0 GHz. Dále na desce můžeme najít 2 GB operační paměti a 8 GB vestavěné úložiště. Tento embedded systém se od ostatních výše zmíněných liší tím, že na desce má osazený audio konektor S/PDIF a DisplayPort, což umožnuje připojení až dvou monitorů a také podporou RAID polí díky rozšiřující kartě.

6.1 Použitá zařízení

K otestování mého programu jsem využil počítače architektury ARM. Jedná se o počítače SolidRun HummingBoard Pro, Raspberry PI 3 Model B a Sinovoip Banana PI BPI-M1. Na těchto zařízeních bude otestován algoritmus a všechny výsledky budou zapsány do tabulky v následující kapitole.

HummingBoard Pro

Disponuje čtyřjádrovým procesorem i.MX6 Dual-core Lite na achitektuře Cortex A9 o frekvenci 1 GHz. Na desce má osazenou paměť typu DDR3 o velikosti 2 GB a nabízí grafický obvod Vivante GC880. Dále deska nabízí mSata konektor pro připojení SSD disku nebo například IR přijímač. Podporuje známé operační systémy Linux, například Android 4.4, Debian a OpenSuse. Spotřeba zařízení je 2 W (0,41 A) v klidovém stavu a 5 W (1 A) v zátěži.

Raspberry PI 3

Jedná se o třetí generaci velmi úspěšné řady Raspberry PI. Model je osazen výkonným 64 bitovým čtyřjádrovým procesorem Cortex-A53 o frekvenci 1,2 Ghz, grafickým čipem Broadcom VideoCore IV o frekvenci 400 MHz a operační pamětí typu SDRAM o velikosti 1 GB, která je sdílená s grafickým čipem. Od svých předchůdců se liší integrovaným Wifi čipem podporující protokoly 802.11 b/g/n a Bluetooth 4.1 LE. Díky architektuře ARMv8 má širší podporu Linuxů, včetně mobilního systému Android a Windows 10 IoT. Jeho výkon bez zátěže je 1,5 W (300 mA) a při zátěži se zapojenými periferiemi maximálně 6,7 W (1,34 A).

Banana PI

Tento počítač je osazen dvoujádrovým procesorem AllWinner A20 s frekvencí 1,2 GHz. Jedná se o low-end verzi procesoru AllWinner A31. Dále na desce najdeme grafický čip Mali-400 MP2 a operační paměť 1 GB. Jedná se o první klon RPI a jeho deska je navíc osazena například sata konektorem, mikro-USB OTG nebo mikrofonem. Tento mikropočítač díky GLAN můžeme použít jako vzdálené NAS úložiště (*Network Attached Storage*), databázi, mail server nebo například web server. Jeho spotřeba v nečinném stavu je 1,75 W (350 mA), maximálně 5.5 W (1,1 A).

6.2 Srovnání použitých zařízení

Jedná se o poměrně výkonné počítače vzhledem ke své velikosti. V tabulce 1 se nachází srovnání některých parametrů těchto počítačů.

Tabulka 1: Srovnání testovaných zařízení

	HummingBoard Pro	Raspberry PI 3	Banana PI
Procesor	NXP i.MX6 ARM	ARM	A20 ARM
Počet jader	4	4	2
Frekvence CPU	1 GHz	1,2 GHz	1,2 GHz
Druh architektury	Cortex A9	Cortex A53	Cortex A7
Instrukční sada	ARMv7-A	ARMv8-A	ARMv7-A
Grafický čip	Vivante GC880	BroadCom VideoCore IV	Mali-400 MP2
Kapacita paměti RAM	1 GB	1 GB	1 GB
Druh paměti	DDR3	LPDDR2	DDR3
Ethernet	10/100/1000	10/100	10/100/1000
Počet USB portů	2	4	2
Úložný prostor	MicroSD	MicroSDHC	SD/MMC
Rok vydání	2014	2016	2014

7 Experimenty

Experimenty této práce probíhaly především na počítačích uvedených v předchozí kapitole. Nejprve je však vhodné zmínit konfiguraci daných počítačů:

- **HummingBoard Pro** - na tomto počítači je nainstalován Linux Debian 8.1 s označením „jessie“ s grafickým rozhraním MATE 8.
- **Banana PI** - na tomto počítači je nainstalován Linux Debian 8.1 s označením „jessie“ s grafickým rozhraním Xfce 4.10.
- **Raspberry PI** - na tomto počítači je nainstalován Linux Raspbian.

Dále pro srovnání výkonu těchto počítačů byl program otestován i na desktopovém počítači, na kterém byl zároveň vyvíjen a odladěn. Jeho parametry jsou následující:

- *Intel Core i7–6700k*, operační paměť *32 GB* a operační systém *Windows 10 Pro*.

První sada experimentů byla detekce pomocí histogramů orientovaných gradientů a jeho vylepšení pomocí algoritmu pro substrakci pozadí při detekci ze statických kamer.

7.1 Nalezení optimálního klasifikátoru

Nalezení optimálního a spolehlivého klasifikátoru je klíčovou částí celého procesu. Zároveň se jedná o velmi komplikovaný a zdlouhavý úkol. Je důležité zvolit ideální trénovací sadu vzorků a k vzhledem zvoleného typu SVM i také jejího jádra. Otestoval jsem varianty C–SVC, ν–SVC, ε–SVR klasifikátorové typy s lineárním a Gaussovým jádrem, přičemž jsem zjistil, že nejlepší sestava pro detekování chodců je kombinace typu ε–SVR s lineárním jádrem, protože úspěšnost detekce byla značně vysoká. Po zvolení správného jádra a typu klasifikátoru je dalším krokem vybrat, jak už bylo zmíněno, trénovací sadu. Vyzkoušel jsem nejrůznější trénovací sady dostupné na internetu, a také jejich kombinace.

Nejlepších výsledků jsem dosáhl s pozitivní trénovací sadou Dase [33], CUHK01 campus [42] a jako negativní sadu jsem použil Daimler–Mono[38], kterou jsem nastříhal na vzorky. Tyto sady jsou součastí přílohy této práce. Následujícím krokem je zvolení správných parametrů trénování klasifikátoru.

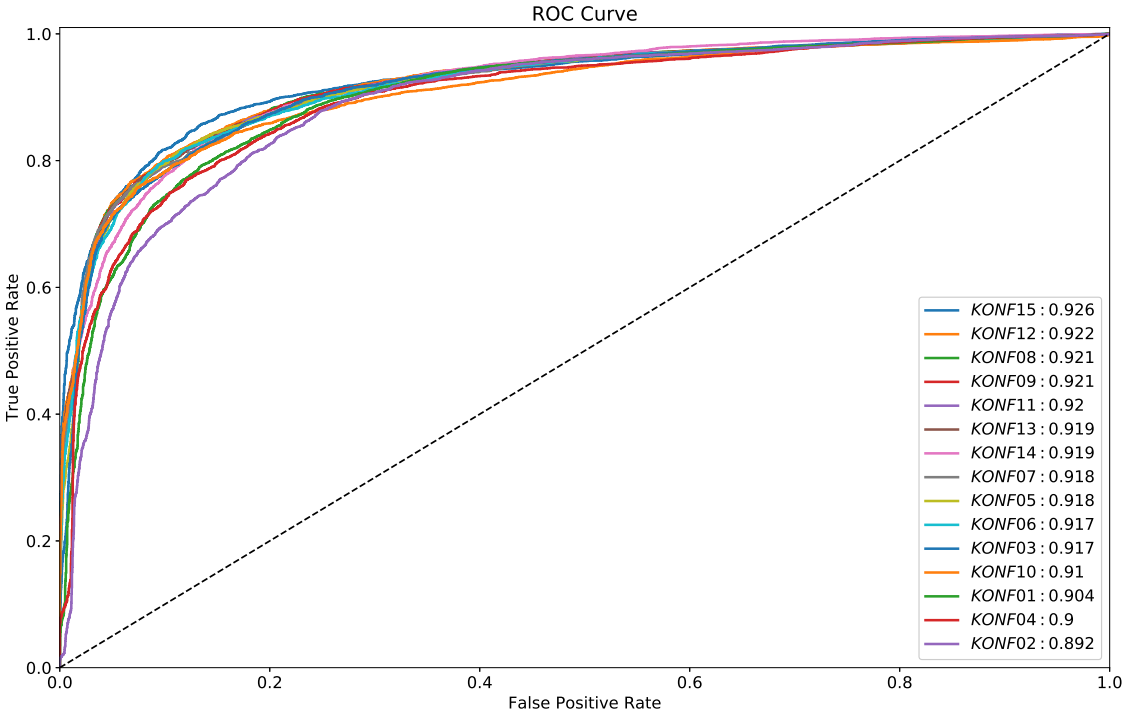
Trénování klasifikátoru je velmi citlivé na tyto parametry. Při změně některého z nich, byť jen o tisícinu, můžeme získat úplně jiný výsledek. Trénování probíhalo dvakrát za sebou. Při druhém trénování se klasifikátor učil ze svých špatných detekcí, čemuž se říká „Bootstrapping“ [45] a tento postup je ilustrován ve zdrojovém kódu 1.

```
        cv::Size posSize = posSamplesLst[0].size();
        cv::HOGDescriptor myHog;
        myHog.winSize = posSize;
        std::vector < cv::Rect > locations;
        std::vector < float > hogDetector;

        getSvmDetector(svm, hogDetector);
        myHog.setSVMDetector(hogDetector);
        std::vector < cv::Rect > detections;

        for (size_t i~= 0; i~< negSamplesLst.size(); i++) {
             myHog.detectMultiScale(negSamplesLst[i], detections);
             for (size_t j = 0; j < detections.size(); j++) {
                  cv::Mat detection = negSamplesLst[i](detections[j]).clone();
                  resize(detection, detection, posSize);
                  negSamplesLst.push_back(detection);
             }
        }
        labels.clear();
        labels.assign(posSamplesLst.size(), +1);
        labels.insert(labels.end(), negSamplesLst.size(), 0);

        gradientLst.clear();
        extractFeatures(posSamplesLst, gradientLst);
        extractFeatures(negSamplesLst, gradientLst);

        convertSamples2Mat(gradientLst, trainMat);
        trainSvm(trainMat, labels);
```

Výpis 1: Bootstrapping

Natrénoval jsem klasifikátory s různými parametry a otestoval je na testovacích vzorcích. Tyto vzorky byly vyextrahované obrázky chodců a "ne-chodců"z testovacího videa pomocí jejich anotací a přiloženého nástroje pro tvorbu vzorků a také z projektu PETA [41], konkrétně se jednalo o sady: *3DPeS, CAVIAR4REID, CBLC, SARC3D a VIPeR*. Tento krok je v podstatě první filtrací klasifikátorů. Vybíráme takový klasifikátor, aby jeho detekce byla co nejvíc přesná a detekoval, co nejméně špatných oblasti. Vytrénoval jsem 15 klasifikátorů s různými parametry trénování. Tyto klasifikátory jsem nadále filtroval pomocí ROC křivky a vybral ten nejlepší podle veličiny AUC z grafu, který je na obrázku 25. Nastavení trénovacích parametrů jsou uvedeny v tabulce 2.

ROC křivka je takový graf, který ilustruje schopnost a optimalizaci binárního klasifikátoru pro všechny přípustné hodnoty prahů. Křivka je vytvořena vynesením relativní četnosti skutečně pozitivních případů (true positive rate - TPR) na svislé ose a relativní četnosti falešně pozitivních případů (false positive rate - FPR) na ose vodorovné. Čím je plocha pod křivkou (AUC) větší, tím má klasifikátor menší chybovost.



Obrázek 25: ROC křivka vytrénovaných klasifikátorů

Tabulka 2: Přehled vytrénovaných klasifikátorů a jejich trénovacích parametrů. V příloze B jsou uvedeny parametry, které konfigurace sdílely mezi sebou.

Název	par. C	par. P	max iterací	počet neg. vzorků	Přesnost %
KONF 01	0.005	0.02	2000	3000	83
KONF 02	0.005	0.01	3500	3000	81
KONF 03	0.001	0.02	2000	6000	83
KONF 04	0.001	0.02	4500	6000	81
KONF 05	0.005	0.4	3500	6000	84
KONF 06	0.01	0.4	3000	6000	84
KONF 07	0.005	0.01	2500	9000	82
KONF 08	0.005	0.25	2000	9000	82
KONF 09	0.005	0.25	3000	9000	82
KONF 10	0.01	0.02	3000	9000	80
KONF 11	0.01	0.25	2000	9000	82
KONF 12	0.01	0.25	2500	9000	82
KONF 13	0.01	0.25	4500	9000	82
KONF 14	0.5	0.1	1200	9000	81
KONF 15	0.5	0.1	2000	9000	83

Vybral jsem konfiguraci 15, jelikož oblast pod křivkou byla největší ze všech vytrénovaných. Přesnost tohoto klasifikátoru je v tabulce 3. Výsledky testování ostatních klasifikátorů jsou v příloze této práce. Detekce může nabývat maximálně 4 stavů:

- True Positive (**TP**) - detektor správně rozpoznal chodce.
- True Negative (**TN**) - detektor tuto oblast správně ignoroval.
- False Positive (**FP**) - detektor označil tuto oblast za chodce, ovšem se ten zde nenachází.
- False Negative (**FN**) - v oblasti se vyskytuje chodec avšak byl ignorován.

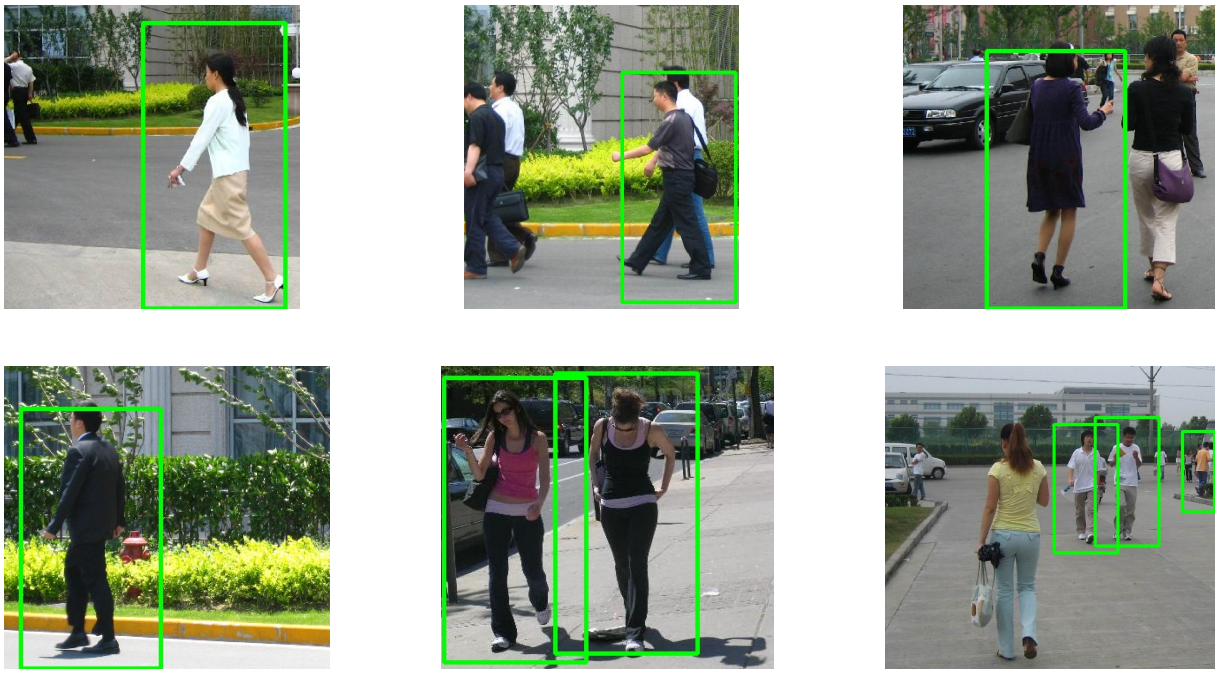
Díky těmto stavům můžeme vypočítat přesnost detekce:

$$Přesnost = \frac{TP + TN}{TP + TN + FP + FN}$$

Tabulka 3: Přesnost konfigurace klasifikátoru číslo 15

	TP/FN	TN/FP	Přesnost %	AUC
KONF 15	3479/1572	4912/192	83	0.926

Na tento klasifikátor jsem se rozhodl aplikovat algoritmus substrakce pozadí, a tak zefektivnit jeho rychlost a správnost detekce na videosekvenci ze statické kamery. Samotný klasifikátor byl také otestován na testovacích obrázcích ze sady [46]. Výsledky detekce jsou na vybraných obrázcích 26. V příloze A jsou umístěny další příklady.

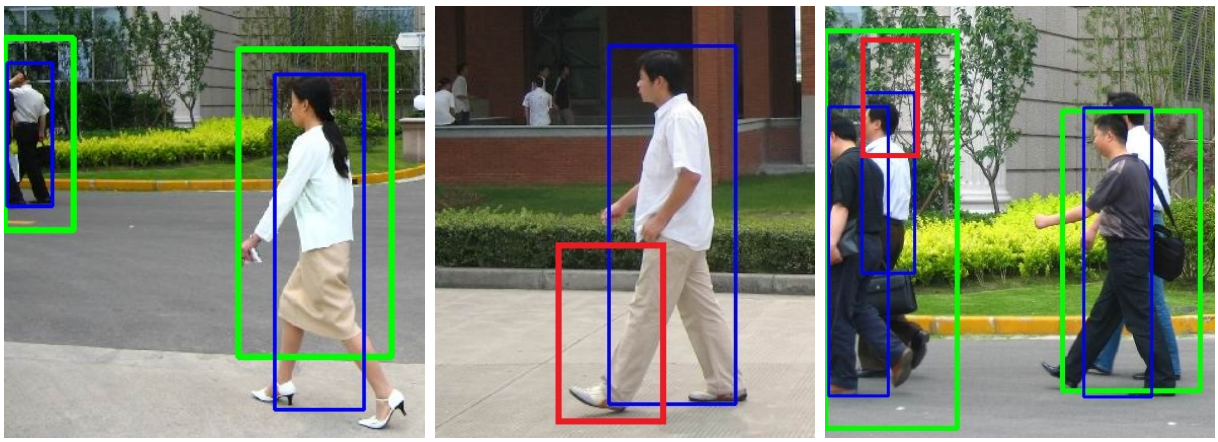


Obrázek 26: Detekce na vybraných testovacích obrázcích. Obrázky jsou v testovací sadě [46].

Chodci se na obrázku mohou vyskytovat kdekoliv, proto se při takové detekci používá výpočet F1 skóre namísto přesnosti. Ke svému výpočtu totiž nepotřebuje true negative, které by bylo obtížné získat z každého snímku. Výpočet F1 skóre je následující:

$$F1\ skóre = \frac{2TP}{2TP + FP + FN}$$

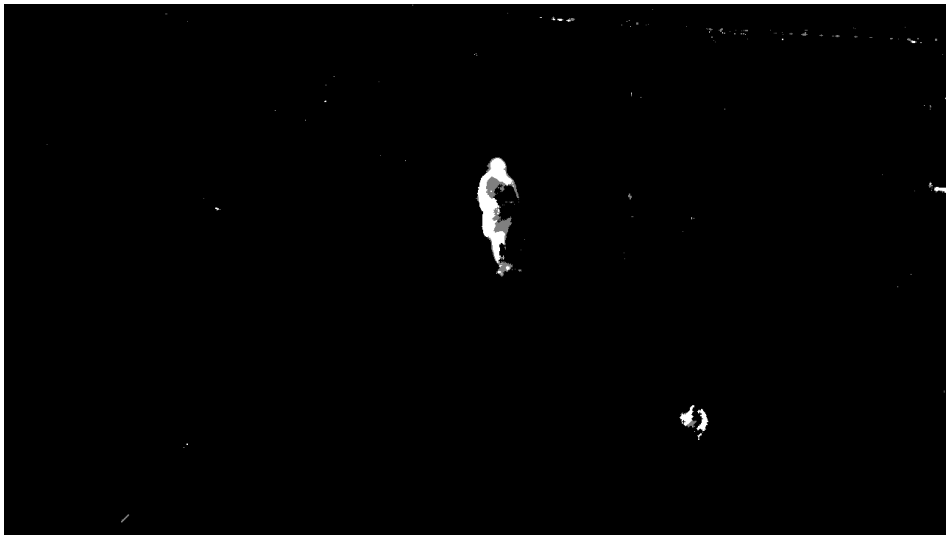
Za správnou detekci je považována taková detekce, kdy výstup z detektoru je alespoň polovina správně anotované oblasti. Příklad správné a chybné detekce je na obrázku 27.



Obrázek 27: Příklady správné (zelené) a chybné (červené) detekce. Modré obdélníky značí anotovanou oblast. Obrázky jsou v testovací sadě [46].

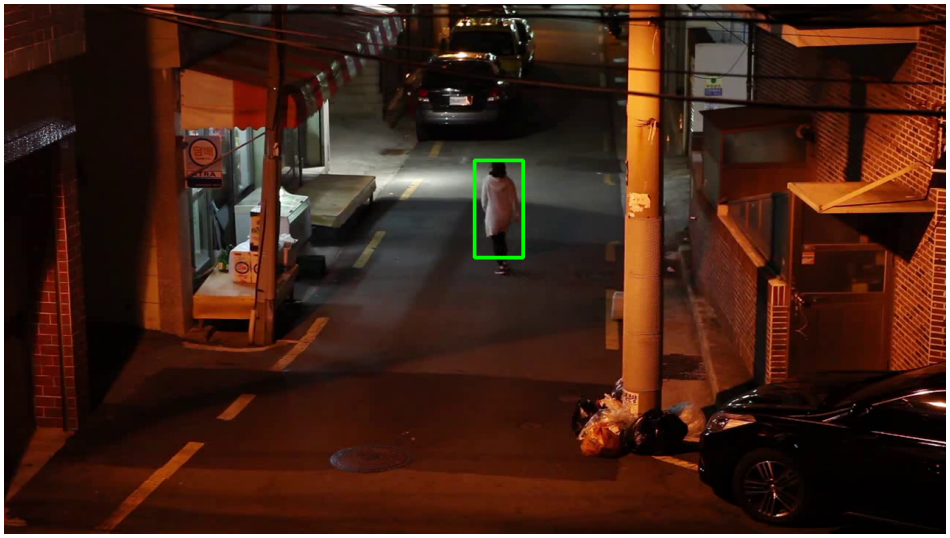
7.2 Aplikace algoritmu Mixtura Gaussiánů

V předchozí kapitole jsem popsal, jak probíhalo zvolení optimálního klasifikátoru a nyní jej použijeme v kombinaci s metodou substrakce pozadí. Detektor nemusí procházet celý obraz, ale soustředí se pouze na vyextrahované části z obrazu. Kombinace metod HOG a MOG může mít významný vliv na výkon celé detekce. Metoda MOG extrahuje celé dynamické popředí, to znamená, že do detektoru se nemusí dostat jenom chodec. Příklad tohoto případu zobrazuje výstupní obrázek 28 metody MOG, kde po extrahování popředí a aplikací eroze a dilatace zůstaly dvě významné oblasti. Tyto dvě oblasti jsou na obrázcích 29. Na obrázku vlevo se nachází hledaný chodec, zatímco na pravém nikoliv. Detekce je spuštěna na obou oblastech, kde chodec byl úspěšně nalezen a druhý výřez byl ignorován. Výsledek detekce je na obrázku 30.



Obrázek 28: Výstupní obrázek z metody MOG.

Obrázek 29: Výřezy z původního obrázku, na kterých je spuštěna detekce.



Obrázek 30: Výsledná detekce, klasifikátor se rozhodnul správně.

Testovací video [47] je ve 3 různých rozlišeních, základní informace o videosekvencích je uvedena v tabulce 4. Chodci v použitých videosekvencích a obrázcích jsou anotováni v textových souborech.

Tabulka 4: Přehled testovaných videozáznamů

Název videa	Rozlišení	Počet snímků	FPS	Výskyt osob na snímcích
cctv4_640	640 × 360	781	29,97	749
cctv4_1280	1280 × 720	781	29,97	749
cctv4_1920	1920 × 1080	781	29,97	749

V tabulkách 5, 6 a 7 jsou uvedeny výsledky detekce z každého videa na jednotlivém zařízení. Tabulka 8 znázorňuje výsledky detekce z desktopového počítače, které slouží jako referenční model k výsledkům výše zmíněných tabulek. Jsou zde také uvedeny výsledky detektoru z knihovny Dlib (dále jen „FHOG“) včetně původního detektoru lidí, který je součástí knihovny OpenCV (dále jen „default HOG“) a slouží k porovnání s vytrénovaným SVM klasifikátorem. Dlib klasifikátor byl vytrénován podle příkladu uvedeného na webové stránce knihovny.

Tabulka 5: Výsledky detekce počítače HummingBoard Pro i.MX6

Název videa	Typ algoritmu	FPS	Délka detekce[s]	TP	FN	FP	F1skóre[%]
cctv4_640	HOG	0,9	838,82	447	302	171	65,34
	HOG + MOG	4,4	176,65	576	173	19	85,71
	FHOG	1,9	404,41	238	511	43	46,21
	FHOG + MOG	**5,8**	**134,10**	347	402	0	63,32
	default HOG	0,9	867,73	299	450	24	55,78
	default HOG + MOG	2,6	306,13	364	385	3	65,23
cctv4_1280	HOG	0,6	1 270,89	492	257	10	78,66
	HOG + HOG	1,2	649,33	606	143	25	**87,83**
	FHOG	0,5	1 651,88	334	415	7	61,28
	FHOG + MOG	1,5	**520,10**	497	252	30	77,9
	default HOG	0,2	4 366,98	719	30	397	77,1
	default HOG + MOG	0,9	897,61	588	161	64	83,94
cctv4_1920	HOG	0,3	2 966,33	573	176	125	79,19
	HOG + MOG	0,3	2 338,31	542	207	66	79,88
	FHOG	0,2	3 756,55	337	412	38	59,96
	FHOG + MOG	0,5	**1 574,09**	355	394	35	62,34
	default HOG	0,1	10 473,00	710	39	670	66,69
	default HOG + MOG	0,5	1 620,12	381	368	94	62,25

Z tabulky 5 je vidět, že metoda HOG s kombinací metodou MOG dosáhla nejvyššího F1 skóre, a to 87,83 % v čase 649 sekund. Bez aplikace metody MOG je délka detekce 1271 sekund a F1 skóre 78,66 %. V tomhle případě tato kombinace nejen zvýšila rychlost detekce, ale i její přesnost. Dále vidíme, že nejrychlejší metoda je FHOG s kombinací MOG pro video s rozlišením 640×360, a to 134 sekund s 5,8 snímky za sekundu. Pro rozlišení 1280×720, respektive 1920×1080 je opět nejrychlejší metoda FHOG s kombinací MOG, a to časem vykonávání 520, respektive 1574 sekund.

Tabulka 6: Výsledky detekce na zařízení Raspberry PI3

Název videa	Typ algoritmu	FPS	Délka detekce[s]	TP	FN	FP	F1 skóre[%]
cctv4_640	HOG	1,2	651,11	450	299	170	65,74
	HOG + MOG	4,1	189,74	576	173	20	85,65
	FHOG	3	262,01	239	510	44	46,32
	FHOG + MOG	**6,8**	**115,27**	347	402	1	63,26
	default HOG	1,1	691,84	299	450	24	55,78
	default HOG + MOG	2,3	339,83	362	387	3	64,99
cctv4_1280	HOG	0,8	980,01	492	257	10	78,66
	HOG + HOG	1,3	609,43	606	143	25	**87,83**
	FHOG	0,7	1 062,23	333	416	7	61 ,16
	FHOG + MOG	1,9	**415,12**	497	252	30	77,9
	default HOG	0,3	2 983,22	719	30	399	77,02
	default MOG + MOG	1	820,78	585	164	69	83,39
cctv4_1920	HOG	0,4	2 083,59	572	177	127	79,01
	HOG + MOG	0,4	1 884,39	545	204	66	80,15
	FHOG	0,3	2 405,86	337	412	38	59,97
	FHOG + MOG	0,8	**1 027,02**	355	394	35	62,34
	default HOG	0,1	6 900,89	710	39	666	66,82
	default HOG + MOG	0,7	1 195,23	380	369	95	62,09

Z tabulky 6 je vidět, že metoda HOG s kombinací metodou MOG dosáhla nejvyššího F1 skóre, a to 87,83 % v čase 609 sekund. Bez aplikace metody MOG je délka detekce 980 sekund a F1 skóre 78,66 %. V tomhle případě tato kombinace nejen zvýšila rychlost detekce, ale i její přesnost. Dále vidíme, že nejrychlejší metoda je FHOG s kombinací MOG pro video s rozlišením 640×360, a to 115 sekund s 6,7 snímky za sekundu. Pro rozlišení 1280×720, respektive 1920×1080 je opět nejrychlejší metoda FHOG s kombinací MOG, a to časem vykonávání 389, respektive 1027 sekund.

Tabulka 7: Výsledky detekce počítače Banana PI BPI-M1

Název videa	Typ algoritmu	FPS	Délka detekce[s]	TP	FN	FP	F1 skóre[%]
cctv4_640	HOG	0,4	1 786,31	446	303	169	65,34
	HOG + MOG	2,9	268,23	575	174	18	85,69
	FHOG	1,7	469,09	238	511	43	46,21
	FHOG + MOG	**6**	**131,03**	347	402	0	63,32
	default HOG	0,4	1 754,02	299	450	24	55,78
	default HOG + MOG	1,4	546,84	364	385	3	65,23
cctv4_1280	HOG	0,3	2 385,32	492	257	10	78,66
	HOG + MOG	0,9	886,61	604	145	28	**87,47**
	FHOG	0,4	1 915,36	334	415	7	61,28
	FHOG + MOG	1,6	**502,80**	497	252	30	77,89
	default HOG	0,1	8 901,18	719	30	397	77,10
	default HOG + MOG	0,5	1 448,49	583	166	69	83,23
cctv4_1920	HOG	0,1	5 654,76	572	177	127	79,01
	HOG + MOG	0,2	3 666,30	543	206	66	79,97
	FHOG	0,2	4 368,47	337	412	38	59,96
	FHOG + MOG	0,5	**1 633,79**	355	394	35	62,34
	default HOG	0,04	21 418,20	710	39	666	66,82
	default HOG + MOG	0,3	2 390,96	380	369	95	62,09

Z tabulky 7 je vidět, že metoda HOG s kombinací metodou MOG dosáhla nejvyššího F1 skóre, a to 87,47 % v čase 887 sekund. Bez aplikace metody MOG je délka detekce 2485 sekund a F1 skóre 78,66 %. V tomhle případě tato kombinace nejen zvýšila rychlost detekce, ale i její přesnost. Dále vidíme, že nejrychlejší metoda je FHOG s kombinací MOG pro video s rozlišením 640×360, a to 131 sekund s 6 snímky za sekundu. Pro rozlišení 1280×720, respektive 1920×1080 je opět nejrychlejší metoda FHOG s kombinací MOG, a to časem vykonávání 503, respektive 1934 sekund.

Tabulka 8: Výsledky detekce desktopového počítače

Název videa	Typ algoritmu	FPS	Délka detekce[s]	TP	FN	FP	F1 skóre[%]
cctv4_640	HOG	19,3	40,54	463	286	136	68,69
	HOG + MOG	96,8	8,07	581	168	15	86,39
	FHOG	32,2	24,25	253	496	33	48,89
	FHOG + MOG	**120,5**	**6,48**	350	399	3	63,52
	default HOG	19,4	40,29	304	445	27	56,3
	default HOG + MOG	53,8	14,52	373	376	5	66,19
cctv4_1280	HOG	12,6	61,87	507	242	9	80,16
	HOG + MOG	31,4	24,84	598	151	31	**86,79**
	FHOG	8,3	94,61	333	416	0	61,55
	FHOG + MOG	35,5	**21,98**	498	251	30	77,99
	default HOG	4	195,47	718	31	298	81,36
	default HOG + MOG	21,9	35,63	583	166	58	83,88
cctv4_1920	HOG	5,5	141,66	560	189	141	77,24
	HOG + MOG	8,3	94,57	569	180	47	83,37
	FHOG	3,6	214,16	336	413	41	59,68
	FHOG + MOG	10,1	77,27	371	378	32	64,41
	default HOG	1,7	472,13	706	43	536	70,92
	default HOG + MOG	12,3	**63,25**	387	362	85	63,39

Z tabulky 8 je vidět, že metoda HOG s kombinací metodou MOG dosáhla nejvyššího F1 skóre, a to 86,79 % v čase 25 sekund. Bez aplikace metody MOG je délka detekce 62 sekund a F1 skóre 80,16 %. V tomhle případě tato kombinace nejen zvýšila rychlost detekce, ale i její přesnost. Dále vidíme, že nejrychlejší metoda je FHOG s kombinací MOG pro video s rozlišením 640×360, a to 6 sekund s 120,5 snímků za sekundu. Pro rozlišení 1280×720 je opět nejrychlejší metoda FHOG s kombinací MOG, a to časem vykonávání 22 sekund. Pro rozlišení 1920×1080 je v tomhle případě nejrychlejší varianta default HOG s kombinací MOG v čase 63 sekund.

Z tabulek můžeme vidět, jak se program choval na každém zařízení jinak. To mohlo být způsobeno různými faktory, například jiným způsobem zaokrouhlování čísel s plovoucí čárkou. Navzdory tomu, že tyto počítače podporují rozšíření NEON a obě knihovny byly zkompilovány s tímto rozšířením, výsledky nebyly tak časově uspokojivé, jak jsem očekával. V tabulce 9 vidíme metody s nejlepší úspěšností detekce na jednotlivém zařízení. Kombinace metod HOG a MOG se prokázala jako nejlepší varianta pro detekci chodců na všech testovaných rozlišení videa. Délka detekce na HummingBoard Pro a Raspberry PI je téměř porovnatelná, na rozdíl od zařízení Banana PI, které se prokázalo jako nevhodné u této metody.

Tabulka 9: Shrnutí na základě úspěšnosti detekce

Název videa	Zařízení	Typ algoritmu	FPS	Délka detekce[s]	F1 skóre[%]
cctv4_640	HummingBoard Pro	HOG + MOG	4,4	176,65	85,71
	Raspberry PI3	HOG + MOG	4,1	189,74	85,65
	Banana PI	HOG + MOG	2,9	268,23	85,69
cctv4_1280	HummingBoard Pro	HOG + MOG	1,2	649,33	87,83
	Raspberry PI3	HOG + MOG	1,3	609,43	87,83
	Banana PI	HOG + MOG	0,9	886,61	87,47
cctv4_1920	HummingBoard Pro	HOG + MOG	0,3	2 338,31	79,88
	Raspberry PI3	HOG + MOG	0,4	1 884,39	80,15
	Banana PI	HOG + MOG	0,2	3 666,30	79,97

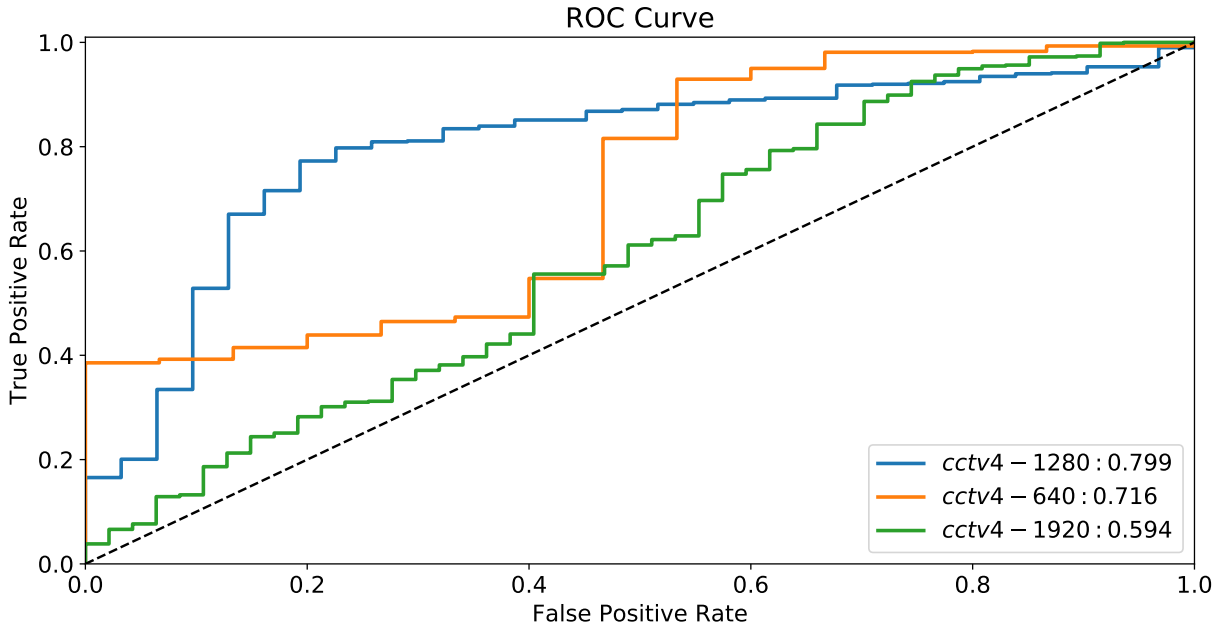
V tabulce 10 vidíme shrnutí nejrychlejších metod pro jednotlivá zařízení. Jednoznačně nejrychlejší metoda se stala kombinace FHOG a MOG na všech zařízeních. Dále pak vidíme, že tato metoda byla nejrychlejší na zařízení Raspberry PI.

Tabulka 10: Shrnutí na základě rychlosti detekce

Název videa	Zařízení	Typ algoritmu	FPS	Délka detekce[s]	F1 skóre[%]
cctv4_640	HummingBoard Pro	FHOG + MOG	5,8	134,10	63,32
	Raspberry PI3	FHOG + MOG	6,8	**115,27**	63,26
	Banana PI	FHOG + MOG	6	131,03	63,32
cctv4_1280	HummingBoard Pro	FHOG + MOG	1,5	520,10	77,9
	Raspberry PI3	FHOG + MOG	1,9	**415,12**	77,9
	Banana PI	FHOG + MOG	1,6	502,80	77.89
cctv4_1920	HummingBoard Pro	FHOG + MOG	0,5	1 574,09	62,34
	Raspberry PI3	FHOG + MOG	0,8	**1 027,02**	62,34
	Banana PI	FHOG + MOG	0,5	1 633,79	62,34

Na obrázku 31 jsou vykresleny ROC křivky všech videí za použití kombinace metod HOG a MOG. Křivky byly vygenerovány obdobně jako při testování klasifikátoru. SVM klasifikátor na každé detekované oblasti provedl pravděpodobnostní zařazení do konkrétní třídy, jinými slovy, jestli se na obrázku nachází chodec nebo nikoliv. V posledním kroku se vygeneroval „ground truth“ soubor k tomuto pravděpodobnostnímu zařazení, za pomocí anotačního souboru. Jak může být zřejmé, klasifikátor není až tak spolehlivý na obsah s vysokým rozlišením(cctv4_1920). Na druhou stranu pro detekci na průměrném rozlišení (cctv4_1280) je spolehlivější. Natrénoval jsem klasifikátor na vzorcích o velikosti 48×96 proto, aby bylo možné detektor použít i na menších obrázcích bez nutnosti jejich zvětšení. Klasifikátor z knihovny Dlib se mi nepodařilo vy-

trénovat na spolehlivou míru přesnosti pro nejmenší testované rozlišení (cctv4_640). Na druhou stranu, tento klasifikátor byl nejrychlejší na všech testovaných ARM zařízeních.



Obrázek 31: ROC křivky klasifikátoru konfigurace 15 na videosekvencích z tabulky 4

Předfiltrování obrazu pomocí subtrakce pozadí se značně projevilo při detekci chodců. Nejen, že se zefektivnila detekce odfiltrováním většiny false positive detekcí, ale také rychlost samotného detektoru. Například na dvoujádrovém počítači Banana Pi je rychlost skoro až 6× rychlejší než se samotnou metodou HOG při nízkém rozlišení videa (cctv04_640).

Nejvhodnějším zařízením pro detekování chodců při této implementaci se projevil Raspberry PI. Navzdory optimalizování algoritmu nelze hovořit o úspěchu, protože touto implementací nelze detekovat chodce v reálném čase.

8 Závěr

Dle zadání práce byly otestovány vybrané metodiky z knihoven OpenCV a Dlib pro detekci chodců. Z těchto metodik jsem vybral histogram orientovaných gradientů v kombinaci s lineárním klasifikátorem SVM z OpenCV a z knihovny Dlib také histogram orientovaných gradientů. Použití těchto algoritmů obsahuje implementovaná aplikace. Lineární klasifikátor byl zvolen nejen díky své rychlosti klasifikace lineární dělící nadrovinou, ale také proto, že toto jádro je podporováno v metodě s posuvným oknem, tím pádem nebylo nutné vytvářet vlastní implementaci této detekce, která by jistě nedosahovala tak velkého výpočetního výkonu.

Při trénování klasifikátoru hrálo důležitou roli správné zvolení trénovací sady a trénovacích parametrů. Trénovací sada by měla být co nejkvalitnější a měla by obsahovat co nejméně stínů a artefaktů. Jak je zmíněno v textu, aplikace disponuje křížovou validací a testováním klasifikátoru, které sloužilo ke zvolení co nejvíce optimálních parametrů a sady tak, abych docílil co nejpřesnějšího a optimálního klasifikátoru.

Zjistil jsem, že má implementace dané problematiky detekce chodců na embedded zařízeních není dostačující. Na druhou stranu metoda substrakce pozadí značně urychlila detekci, a to minimálně na jeden snímek za sekundu.

Aplikace by mohla být v budoucnu obohacena o vlastní implementaci substrakce pozadí nebo jiným způsobem ořezání výstupního obrazu této substrakce, což by mohlo značně zvýšit výkon aplikace. Dále by práce mohla být rozšířena o další klasifikátory, například kaskádovými. Tyto kaskádové klasifikátory by značně zvýšily výkon detekce chodců na daných zařízeních a dosahovaly by většího počtu snímků za sekundu. Dalším možných vývojem je zvolit ARM zařízení s grafickým jádrem a implementovat histogram orientovaných gradientů za pomocí technologie Cuda. Také předběžné zpracování obrazu před samotnou detekcí by mohlo být provedeno paralelně.

Literatura

[1] *Dostupné z: http://humanorigins.si.edu/sites/default/files/styles/home_slider_phablet/public/KidComp_landscape.jpg*

[2] SKLANSKY, Jack. *Finding the convex hull of a simple polygon.* [online] Pattern Recognition Letters, December 1982 . [cit. 11.03.2018]. *Dostupné z: http://www.sciencedirect.com/science/article/pii/0167865582900162*

[3] HAMID, Umar, Zakir, Abdul, ZAKUAN, Fakhrul, Razi, Ahmad, ZULKEPLI, Khairul, Akmal, AZMI, Muhammad, Zulfaqar, ZAMZURI, Hairi, RAHMAN, Mohd, Azizi, Abdul, ZAKARIA, Muhammad, Aizzat, *Autonomous emergency braking system with potential field risk assessment for frontal collision mitigation.* [online] IEEE, 2017 [cit. 10.03.2018]. *Dostupné z: https://ieeexplore.ieee.org/document/8313024/*

[4] ZIVKOVIC, Zoran. *Improved adaptive Gausian mixture model for background subtraction.* [online] International Conference Pattern Recognition, UK, August, 2004 [cit. 11.03.2018]. *Dostupné z: http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf Dostupné z: http://ieeexplore.ieee.org/document/4359319/*

[5] ITSEEZ, Open Source Computer Vision documentation v3.2.0 *How to Use Background Subtraction Methods.* [online] Itseez, April, 2014 [cit. 11.03.2018]. *Dostupné z: http://docs.opencv.org/3.2.0/d1/dc5/tutorial_background_subtraction.html*

[6] DALAL, Navneet, TRIGGS, Bill. *Histogram of oriented gradients for human detection.* [online] IEEE, July 25, 2005. [cit. 11.03.2018]. *Dostupné z: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf*

[7] BELONGIE, Serge, MALIK, Jitendra, *Matching with shape contexts.* [online] IEEE, August 06, 2002 [cit. 30.03.2018]. *Dostupné z: http://ieeexplore.ieee.org/document/853834/*

[8] LOWE, David, *Object recognition from local scale-invariant features.* [online] IEEE, September, 1999 [cit. 30.03.2018]. *Dostupné z: http://ieeexplore.ieee.org/document/790410/*

[9] *Dostupné z: http://www.learnopencv.com/wp-content/uploads/2016/11/hog-cells.png*

[10] CHANG, Chih-Chung, LIN, Chin-Jen, *LIBSVM: A Library for Support Vector Machines.* [online] ACM Transactions on Intelligent Systems and Technology, 2001 [cit. 31.03.2018]. *Dostupné z: https://www.csie.ntu.edu.tw/~cjlin/libsvm/*

[11] KARATZOGLOU, Alexandros, SMOLA, Alex, HORNIK, Kurt, ZEILEIS, Achim *kernlab – An S4 Package for Kernel Methods in R.* [online] Journal of Statistical Software, 2004 [cit. 31.03.2018]. *Dostupné z:*

[12] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., DUCHESNAY, E., *Scikit-learn: Machine Learning in Python.* [online] Journal of Machine Learning Research, 2011 [cit. 31.03.2018]. *Dostupné z: http://support.sas.com/documentation/cdl/en/whatsnew/64209/HTML/default/viewer.htm#emdocwhatsnew71.htm*

[13] JOACHIMS, Thorsten *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning.* [online] MIT-Press, 1999 [cit. 31.03.2018]. *Dostupné z: http://svmlight.joachims.org/*

[14] VAPNIK, Vladimir, CORTES, Corinna, *Support-vector networks.* [online] Kluwer Academic Publishers, February 20, 1995 [cit. 30.03.2018]. *Dostupné z: https://link.springer.com/article/10.1007%2FBF00994018*

[15] KOWALCZYK, Alexandre. *Support Vector Machines Succinctly.* [online] Syncfusion, Published on October 23, 2017 [cit. 11.03.2018]. *Dostupné z: https://www.syncfusion.com/ebooks/support_vector_machines_succinctly*

[16] BOSER, Bernhard, GUYON, Isabelle, VAPNIK, Vladimir, *A training algorithm for optimal margin classifiers.* [online] ACM Press, 1992 [cit. 01.04.2018]. *Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.3818*

[17] SCHÖLKOPF, Bernhard, SMOLA, Alex, WILLIAMSON, Robert, BARTLETT, Peter, *New support vector algorithms.* [online] MITP, Neural Computation, May 1, 2000 [cit. 01.04.2018]. *Dostupné z: http://ieeexplore.ieee.org/document/6790203/*

[18] SCHÖLKOPF, Bernhard, PLATT, John, SHAWE-TAYLOR, John, SMOLA, Alex, WILLIAMSON, Robert, *Estimating the support of a high-dimensional distribution.* [online] MITP, Neural Computation, July 1, 2001 [cit. 01.04.2018]. *Dostupné z: http://ieeexplore.ieee.org/document/6790022/*

[19] VAPNIK, Vladimir *Fast training of support vector machines using sequential minimal optimization.*. Wiley, New York, NY, 1998.

[20] HOANG, Van-Dung, VAVILIN, Andrey, JO, Kang-Hyun, *Pedestrian detection approach based on modified Haar-like features and AdaBoost.* [online] IEEE, October, 2012 [cit. 31.03.2018]. *Dostupné z: http://ieeexplore.ieee.org/document/6393256/*

[21] VIOLA, Paul, JONES, Michael, *Robust real-time face detection.* [online] IEEE, August 07, 2002 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/937709/

[22] HE, Dong-chen, WANG, Li, *Texture Unit, Texture Spectrum, And Texture Analysis.* [online] IEEE, July, 1990 [cit. 02.04.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/572934/

[23] OJALA, T, PIETIKAINEN, M, HARWOOD, D, *Performance evaluation of texture measures with classification based on Kullback discrimination of distributions.* [online] ICPR, Proceedings of the 12th IAPR International Conference on Pattern Recognition, 1994 [cit. 02.04.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/576366/

[24] AHONEN, Timo, HADID, Abdenour, PIETIKAINEN, Matti, *Face Description with Local Binary Patterns: Application to Face Recognition.* [online] IEEE, October 30, 2006 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/1717463/

[25] FREUND, Yoav, E. SCHAPIRE, Robert, *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting.* [online] Journal of Computer and System Sciences vol. 55, 1997 [cit. 01.04.2018]. *Dostupné z:* https://www.sciencedirect.com/science/article/pii/S002200009791504X

[26] WANG, Xiaoyu, HAN, Tony X, YAN, Shuicheng, *An HOG-LBP human detector with partial occlusion handling.* [online] IEEE, July 29, 2010 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/5459207/

[27] RAUF, Rabia, SHAHID, Ahmad, R, ZIAUDDIN, Sheikh, SAFI, Asad, Ali, *Pedestrian detection using HOG, LUV and optical flow as features with AdaBoost as classifier.* [online] IEEE, January 19, 2010 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/7821024/

[28] CHENG, Yizong, *Mean shift, mode seeking, and clustering.* [online] IEEE, August, 1995 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/400568/

[29] MEER, P, COMANICIU, D, *Mean shift: a robust approach toward feature space analysis.* [online] IEEE, August 27, 2002 [cit. 31.03.2018]. *Dostupné z:* http://ieeexplore.ieee.org/document/1000236/

[30] *Dostupné z:* https://cdn-images-1.medium.com/max/800/1*ZX05x1xYgaVoa4Vn2kKS9g.png

[31] HILTZ, Frederick, *Analog Computer Simulation of a Neural Element.* [online] IEEE, January, 1962 [cit. 16.04.2018]. *Dostupné z:* https://ieeexplore.ieee.org/document/4322944/

[32] LE CUN, Yann, JACKEL Lawrence, BOSER Bernhard, DENKER John, GRAF, Hans, GUYON, Isabelle, HENDERSON, William, HOWARD, Richard, HUBBARD, Douglas, *Handwritten digit recognition: applications of neural network chips and automatic learning.* [online] IEEE, November, 1989 [cit. 16.04.2018]. *Dostupné z: https://ieeexplore.ieee.org/document/41400/*

[33] Dostupné z: https://www.kaggle.com/sudipdas/datasets

[34] Dostupné z: http://pascal.inrialpes.fr/data/human/

[35] Dostupné z: https://data.vision.ee.ethz.ch/cvl/aess/

[36] Dostupné z:
https://www.mpi-inf.mpg.de/departments/computer-vision-and-multimodal-computing/research/people-detection-pose-estimation-and-tracking/multi-cue-onboard-pedestrian-detection/

[37] Dostupné z: http://www.cvlibs.net/datasets/kitti/eval_object.php

[38] Dostupné z: http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/Daimler_Mono_Ped__Detection_Be/daimler_mono_ped__detection_be.html

[39] Dostupné z: http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/Daimler_Stereo_Ped__Detection_/daimler_stereo_ped__detection_.html

[40] Dostupné z: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/datasets/USA/

[41] Dostupné z: http://mmlab.ie.cuhk.edu.hk/projects/PETA.html

[42] Dostupné z: http://www.ee.cuhk.edu.hk/~xgwang/CUHK_identification.html

[43] ITSEEZ *Open Source Computer Vision Library.* [online] 2015 [cit. 14.04.2018]. *Dostupné z: http://opencv.org/*

[44] KING, E, Davis, *Dlib-ml: A Machine Learning Toolkit.* [online] Journal of Machine Learning Research, 2009 [cit. 14.04.2018]. *Dostupné z: http://jmlr.csail.mit.edu/papers/volume10/king09a/king09a.pdf*

[45] BEYELER, Michael, *Machine Learning for OpenCV.* [online]. Birmingham, Packt Publishing, 2017 [cit. 20.3.2018]. *Dostupné z: http://public.eblib.com/choice/publicfullrecord.aspx?p=4917219*

[46] *Dostupné z: https://www.cis.upenn.edu/~jshi/ped_html/*

[47] *Dostupné z: https://www.amazon.com/clouddrive/share/YfQqz4nA33KW4OsvdFN8uHvhnUjWU0KeH2rtOYIsmIj*

A Ukázky detekce chodců

Detekce ve videosekvenci s konfigurací 15

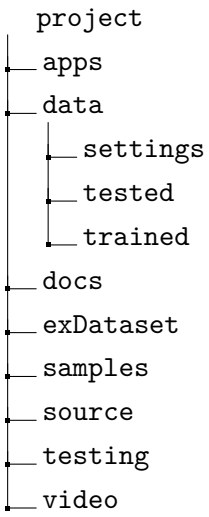
Testovací video je z testovací sady [47].

Detekce na obrázcích s konfigurací 15

Obrázky jsou z testovací sady [46].

B Obsah přiloženého CD

Struktura adresáře



Součastí jsou i vytrénované klasifikátory z knihovny OpenCV (KONF_15.yml) a Dlib (pedDet.svm).

Apps Ve složce apps jsou umístěné skripty pro generování ROC křivek, textové soubory s pravděpodobnostním zařazením a výsledky křížové validace, resp. testování klasifikátoru.

Data Zde se nacházejí soubory s nastavením, anotační soubory videí a obrázků a složku tested, kde se ukládá výstup detektoru.

Docs Zde je umístěna programátorská dokumentace. Spouštění se provádí skrze soubor s názvem index.html.

ExDataset Zde jsou ukázky použitých datasetů.

Samples Zde jsou soubory s cestami k negativním a pozitivním vzorkům.

Source Zde se nachází zdrojový kód implementované aplikace.

Testing Ve složce se nachází testovací soubor k otestování všech kombinací metod a výběr testovacích obrázků ze sady [46].

Video Zde jsou umístěné videosekvence určené k testování.

Uživatelská příručka

Stručnou nápovědu k této aplikaci také získáme spuštěním s argumentem **-h** nebo **-help**.

Spuštění detekce

Aplikace dokáže detekovat ze tří druhů zdrojů, z videa, obrázků a webové kamery. Detekce ve videosekvenci se spustí argumentem **-v=<cesta-k-souboru>**, detekce z webové kamery **-c=<číslo-zařízení>** a z obrázků **-i=<cesta-k-souboru-s-obrázky>**. Spuštění detekce s vlastním detektorem vyžaduje další argumenty a to **-class=<cesta-ke-klasifikátoru>** a pro soubor s jeho nastavením **-st=<cesta-k-souboru>**. Argumentem **-viz=1** povolíme náhled detekce. Například spuštění detekce může vypadat takto:

```
./Pedestrian -v=video/cctv4_640.mp4 -class=KONF_15.yml
   -settings=data/settings/settings_640.txt -viz=1
```

Trénování klasifikátoru

Trénování spustíme argumentem **-type=train**. Po spuštění aplikace s tímto argumentem bude uživatel vyzván, aby vybral typ trénování. Nastavení pro trénování se nachází externě mimo program v souboru. Pokud aplikaci nespustíme s argumentem **-st=<cesta-k-souboru>**, načte se nastavení z výchozího souboru *settings.txt*. Argumentem **-verbose=1** vytiskneme aktuální nastavení trénovacích parametrů před trénovacím procesem.

Testování, křížová validace klasifikátoru

Aplikace také disponuje testováním a křížovou validací klasifikátoru, tu spustíme argumentem **-type=test**. Uživatel následně je vyzván k výberu testování. Aplikace zvládá křížovou validaci klasifikátorů z knihovny OpenCV i z knihovny Dlib. Pro testování nabídne pouze klasikátor z knihovny OpenCV.

Tvorba negativních vzorků

Nástroj se spouští argumentem **-cs=<cesta-k-souboru-se-snímky>**. Po spuštění začne okamžitě pracovat. Snímky ukládá do složky „makedSamples“ a po dokončení vypíše do konzole dobu trvání procesu. Velikost okna se definuje v externím souboru nastavení, pokud chceme definovat velikost řezu, musíme nástroj spustis společně s **-st=<cesta-k-souboru>**.

Anotační nástoj

Nástroj se spouští argumentem **-e=<název-videa>**, po spuštění je uživatel dotázán, kolik osob bude anotovat (nanejvýš lze až 5 osob). Klávesami ‘0’ až ‘4’ se přepíná mezi anotacemi,

přičemž se tato selekce vypíše pro kontrolu do konzole. Klávesa ‘r’ slouží pro vynulování vybrané anotace. Klávesou ‘n’ přeskočíme aktuální snímek bez uložení anotovaných pozic. Klávesa ‘s’ uloží aktuální anotované pozice do paměti a přepne se na další snímek, přičemž pozice první anotace zůstane nezměněná, to umožňuje jednodušší práci, pokud se v obraze nachází pouze jeden chodec. Klávesy ‘i’, ‘j’, ‘k’ a ‘l’ umožňuje pohyb v obraze, pro přesnější umístění anotované části. Pokud tyto klávesy stiskneme společně s klávesou ‘shift’, umožní nám to měnit velikost aktuálně vybrané anotace. Klávesa ‘x’ uloží aktuální pozice anotací v paměti do souboru. Tato funkcionalita umožňuje přerušovanou práci na anotacích videosekvence, ale vyžaduje manuální úpravu souboru uživatelem. Po posledním snímku z videa se nástroj ukončí a soubor s anotacemi se uloží na disk se stejným názvem jako bylo zpracovávané video. Soubor je kompatibilní s jakýmkoliv textovým editorem.

Nastavení trénovacích parametrů

Sdílené trénovací parametry pro všechny konfigurace.

typ (type): EPS_SVR
jádro (kernel) : $LINEAR$
ε: $1 \cdot 10^{-3}$
kritéria ukončení (termCriteria): $CV\_TERMCRIT\_ITER + CV\_TERMCRIT\_EPS$
pozitivních vzorky ze souboru: *sudipDas.txt* (celkem 5649)